CONTROL DATA CYBER 180
MAINFRAME
MODEL-INDEPENDENT
GENERAL DESIGN SPECIFICATION

Doc. No. ARH1700

Rev. AE

BCCB approved December 19, 1989:

T. C. Boos

T. R. Kirsch

H. T. Koppen

E. H. Michehl

E. F. Morris

T. J. Riley

W. B. Williamson

D. M. Powers

Record of Revisions

A - G (03/12/76)	Revisions A through G of this document are identical to revisions A through G of the Integrated Product Line (IPL) Processor-Memory Model-Independent General Design Specification, Doc. No. ASL00211, which is now obsolete.
H - Y (12/18/85)	Revisions H through Y incorporated changes approved between March 1976 and December 1985. Contact AD&C (ARH293) for a detailed listing of these changes.
Z (06/01/86)	This revision incorporates DAP Nos. ARH6723, ARH7085, ARH7185, ARH7195, and ARH7248; and miscellaneous editorial and clarification changes.
AA (03/18/87)	This revision incorporates DAP Nos. ARH7251, ARH7266, ARH7269, ARH7402 ARH7425, ARH7426, ARH7525, ARH7571, ARH7611, ARH7612, and ARH7648; and miscellaneous editorial and clarification changes.
AB (12/18/87)	This revision incorporates DAP Nos. ARH7569, ARH7570, ARH7613, ARH7645, ARH7657, ARH7665, ARH7822, ARH7836, ARH7837, ARH7839, ARH7847, ARH7849, and ARH7899; and miscellaneous editorial and clarification changes.
AC (07/15/88)	This revision incorporates DAP Nos. ARH7855, ARH7987, ARH7996, ARH8108, ARH8259, ARH8260, ARH8261, ARH8267, and ARH8268; and miscellaneous editorial and clarification changes. This edition obsoletes all previous editions.
AD (09/01/89)	This revision incorporates DAP Nos. ARH8290, ARH8391, ARH8510, ARH8728, and ARH8786; and miscellaneous editorial and clarification changes. Due to strategy changes, approved DAP Nos. ARH8257, ARH8258, and ARH8262 were *not* incorporated in this revision; similarly, approved DAP Nos. ARH8259, ARH8260, ARH8261, ARH8267, and ARH8268 (previously incorporated in Rev. AC) were withdrawn from the MIGDS, removing all references to the I4CC and 940 systems.
AE (12/19/89)	This revision incorporates DAP Nos. ARH8732, ARH8877, ARH8895, and ARH8910; and miscellaneous editorial and clarification changes.

Table of Contents

1.0 INTRODUCTION 1-1
1.1 SCOPE 1-1
1.2 APPLICABLE DOCUMENTS 1-1
1.2.1 Control Documents 1-1
1.2.2 Reference Documents 1-1
1.3 CONFIGURATIONS 1-2
1.3.1 Interelement Transfer Paths 1-2
1.3.2 Interelement Clock 1-2
1.3.3 Interelement Connection Alternatives 1-2
1.4 GENERAL TIMING CONSIDERATIONS 1-3
1.5 SYSTEM ELEMENTS 1-4
1.5.1 Element Identifiers (EID) 1-4
1.5.2 Options Installed (OI) 1-5
1.5.3 Microcode Naming Convention 1-5
1.5.4 Configuration Guidelines 1-6
1.5.5 Systems Supported 1-7
1.6 MODEL DIFFERENCES 1-10

2.0 PROCESSOR 2-1
2.1 GENERAL DESCRIPTION 2-1
2.1.1 General Registers 2-1
2.1.1.1 P Register 2-2
2.1.1.2 A Registers 2-2
2.1.1.3 X Registers 2-3
2.1.2 Programming Restrictions 2-3
2.1.3 Instructions 2-4
2.1.3.1 Formats jkiD and SjkiD 2-4
2.1.3.2 Format jk 2-4
2.1.3.3 Format jkQ 2-4
2.1.3.4 P Address Access 2-5
2.1.3.5 Unused Bits 2-7
2.1.3.6 Nomenclature 2-9
2.1.4 Address Arithmetic 2-10
2.1.5 Address Exception 2-10
2.1.6 Instruction Reference Numbers 2-10
2.1.7 Zero Field Length 2-10
2.2 GENERAL INSTRUCTIONS 2-11
2.2.1 Load and Store 2-11
2.2.1.1 Load/Store Bytes, Xk; Length Per S 2-11
2.2.1.2 Load/Store Word, Xk 2-12
2.2.1.3 Load/Store Bytes, Xk; Length Per X0 2-13
2.2.1.4 Load Bytes Xk; Length Per j 2-13
2.2.1.5 Load/Store Bit, Xk 2-14
2.2.1.6 Load/Store Address, Ak 2-15
2.2.1.7 Load/Store Multiple 2-16

2.2.2 Integer Arithmetic 2-19
2.2.2.1 Integer Sum, Xk 2-20
2.2.2.2 Integer Difference, Xk 2-20
2.2.2.3 Integer Product, Xk 2-21
2.2.2.4 Integer Quotient, Xk 2-21
2.2.2.5 Half Word Integer Sum, XkR 2-22
2.2.2.6 Half Word Integer Difference, XkR 2-22
2.2.2.7 Half Word Integer Product, XkR 2-23
2.2.2.8 Half Word Integer Quotient, XkR 2-23
2.2.2.9 Integer Compare 2-24
2.2.3 Branch 2-24
2.2.3.1 Conditional, X 2-25
2.2.3.2 Conditional, X Right 2-25
2.2.3.3 Branch and Increment 2-26
2.2.3.4 Branch on Segments Unequal 2-26
2.2.3.5 Branch Relative 2-27
2.2.3.6 Intersegment Branch 2-27
2.2.4 Copy 2-28
2.2.4.1 Copy, Xk Replaced by Xj 2-28
2.2.4.2 Copy, Xk Replaced by Aj 2-28
2.2.4.3 Copy, Ak Replaced by Aj 2-28
2.2.4.4 Copy, Ak Replaced by Xj 2-28
2.2.4.5 Copy, XkR Replaced by XjR 2-28
2.2.5 Address Arithmetic 2-29
2.2.5.1 Address Increment, Signed Immediate 2-29
2.2.5.2 Address Relative 2-29
2.2.5.3 Address Increment, Indexed 2-29
2.2.5.4 Address Increment, Modulo 2-30
2.2.6 Enter 2-30
2.2.6.1 Enter Immediate 2-30
2.2.6.2 Enter Xk, Signed Immediate 2-30
2.2.6.3 Enter X0 or X1, Immediate Logical 2-31
2.2.6.4 Enter Signs 2-31
2.2.6.5 Enter X0 or X1, Signed Immediate 2-31
2.2.7 Shift 2-32
2.2.7.1 Shift Circular 2-33
2.2.7.2 Shift End-Off 2-33
2.2.8 Logical 2-34
2.2.8.1 Logical Sum, Difference, and Product 2-34
2.2.8.2 Logical Complement 2-34
2.2.8.3 Logical Inhibit 2-35
2.2.9 Register Bit String 2-35
2.2.9.1 Isolate Bit Mask 2-36
2.2.9.2 Isolate 2-36
2.2.9.3 Insert 2-36
2.2.10 Mark to Boolean 2-37

2.3 BUSINESS DATA PROCESSING INSTRUCTIONS 2-38
2.3.1 General Description 2-38
2.3.1.1 Operation Codes 2-39
2.3.1.2 Access Types 2-39
2.3.1.3 Undefined Results for Invalid BDP Data 2-40
2.3.1.4 Overlap 2-40
2.3.2 Data Descriptors 2-40
2.3.2.1 Data Descriptor Interpretation 2-40
2.3.2.1.1 BDP Operand Address, 0 Field 2-40
2.3.2.1.2 BDP Operand Type, T Field 2-41
2.3.2.1.3 BDP Operand Length, F and L Fields 2-42
2.3.2.2 Data and Sign Conventions 2-43
2.3.3 BDP Numeric 2-46
2.3.3.1 Arithmetic 2-47
2.3.3.2 Shift 2-49
2.3.3.3 Move 2-51
2.3.3.4 Comparison 2-52
2.3.4 Byte 2-52
2.3.4.1 Comparison 2-52
2.3.4.2 Byte Scan 2-54
2.3.4.3 Translate 2-54
2.3.4.4 Move 2-55
2.3.4.5 Edit 2-55
2.3.5 Calculate Subscript 2-61
2.3.6 Immediate Data 2-62
2.3.6.1 Move Immediate Data, D(Ak) replaced by XiR plus D per j 2-62
2.3.6.2 Compare Immediate Data, XiR plus D to D(Ak) per j, result to X1R 2-63
2.3.6.3 Add Immediate Data, D(Ak) replaced by D(Ak) plus XiR plus D per j 2-64
2.4 FLOATING POINT INSTRUCTIONS 2-65
2.4.1 Format: 64-Bit 2-65
2.4.1.1 Standard Numbers 2-66
2.4.1.1.1 Z3 2-67
2.4.1.1.2 N 2-67
2.4.1.2 Nonstandard Numbers 2-68
2.4.1.2.1 ±Z1, ±Z2 2-68
2.4.1.2.2 ±INF 2-68
2.4.1.2.3 ±INDEF 2-68
2.4.1.3 Exponent Arithmetic 2-70
2.4.1.4 Normalization 2-70
2.4.1.5 Exceptions 2-70
2.4.1.6 Double Precision Register Designators 2-70
2.4.1.7 User Mask Bits 2-71
2.4.1.8 Conversion (Int/FP) 2-71
2.4.1.8.1 Convert from Integer to Floating Point 2-71
2.4.1.8.2 Convert from Floating Point to Integer 2-72

2.4.1.9 Arithmetic 2-73
2.4.1.9.1 Floating Point Sum/Difference 2-73
2.4.1.9.2 Floating Point Product 2-76
2.4.1.9.3 Floating Point Quotient 2-77
2.4.1.9.4 Double Precision Floating Point Sum/Difference 2-79
2.4.1.9.5 Double Precision Floating Point Product 2-82
2.4.1.9.6 Double Precision Floating Point Quotient 2-84
2.4.1.10 Branch 2-86
2.4.1.10.1 Compare and Branch 2-87
2.4.1.10.2 Exception Branch 2-88
2.4.1.11 Compare 2-89
2.4.1.12 Results 2-90
2.4.2 Format: 32-Bit 2-102
2.5 LOGICAL ENVIRONMENT 2-102
2.5.1 Processor State Registers 2-102
2.5.1.1 Job Process State (JPS) 2-103
2.5.1.2 Monitor Process State (MPS) 2-103
2.5.1.3 Page Table Address (PTA) 2-103
2.5.1.4 Page Table Length (PTL) 2-104
2.5.1.5 Page Size Mask (PSM) 2-104
2.5.1.6 Element Identifier (EID) 2-104
2.5.1.7 System Interval Timer (SIT) 2-105
2.5.1.8 Processor Identifier (PID) 2-105
2.5.1.9 Virtual Machine Capability List (VMCL) 2-105
2.5.1.10 Keypoint Buffer Pointer (KBP) 2-105
2.5.2 Process State Registers 2-106
2.5.2.1 Program Address Register (P) 2-109
2.5.2.2 A Registers 2-109
2.5.2.3 X Registers 2-109
2.5.2.4 Not Assigned 2-109
2.5.2.5 Flags 2-109
2.5.2.6 User Mask (UM) 2-110
2.5.2.7 Monitor Mask (MM) 2-110
2.5.2.8 User Condition Register (UCR) 2-111
2.5.2.9 Monitor Condition Register (MCR) 2-111
2.5.2.10 Debug Mask (DM) 2-111
2.5.2.11 Keypoint Mask (KM) 2-111
2.5.2.12 Keypoint Code (KC) 2-111
2.5.2.13 Process Inverval Timer (PIT) 2-111
2.5.2.14 Base Constant (BC) 2-112
2.5.2.15 Model-Dependent Flags (MDF) 2-112
2.5.2.16 Segment Table Length (STL) 2-112
2.5.2.17 Untranslatable Pointer (UTP) 2-112
2.5.2.18 Segment Table Address (STA) 2-113
2.5.2.19 Last Processor Identification (LPID) 2-113
2.5.2.20 Trap Enables (TE) 2-113
2.5.2.21 Trap Pointer (TP) 2-113
2.5.2.22 Debug Index (DI) 2-113
2.5.2.23 Debug List Pointer (DLP) 2-114
2.5.2.24 Top of Stack (TOS) 2-114
2.5.2.25 Model-Dependent Word (MDW) 2-114

2.5.2.26 Virtual Machine Identifier (VMID) 2-114
2.5.2.27 Untranslatable Virtual Machine Identifier (UVMID) 2-114
2.5.2.28 Largest Ring Number (LRN) 2-114
2.5.2.29 Sample Program Instruction (SPI) Identifier 2-114
2.5.3 Timers 2-115
2.5.3.1 Process Interval Timer 2-115
2.5.3.2 System Interval Timer 2-115
2.5.4 Stacks 2-116
2.5.4.1 Stack Frames 2-116
2.5.5 Binding Section Segment 2-119
2.5.5.1 Code Base Pointer 2-119
2.5.6 Virtual Machine 2-120
2.5.7 System Deadstart 2-120
2.6 SYSTEM INSTRUCTIONS 2-121
2.6.1 Nonprivileged System Instructions 2-122
2.6.1.1 Program Error 2-122
2.6.1.2 Call Indirect 2-122
2.6.1.3 Call Relative 2-125
2.6.1.4 Return 2-127
2.6.1.5 Pop 2-129
2.6.1.6 Exchange 2-132
2.6.1.7 Keypoint 2-133
2.6.1.8 Compare Swap 2-134
2.6.1.9 Test and Set Bit 2-136
2.6.1.10 Test and Set Page 2-137
2.6.1.11 Copy Free Running Counter 2-137
2.6.1.12 Execute Algorithm 2-137
2.6.1.13 Unimplemented Instructions - Reserved Op Codes 2-138
2.6.1.14 Scope Loop Sync 2-138
2.6.1.15 Purge SFSA Pushdown 2-138
2.6.2 Local Privileged Mode 2-139
2.6.2.1 Load Page Table Index 2-139
2.6.3 Global Privileged Mode 2-141
2.6.3.1 Processor Interrupt 2-141
2.6.4 Monitor Mode 2-142
2.6.5 Mixed Mode 2-142
2.6.5.1 Branch on Condition Register 2-142
2.6.5.2 Copy 2-144
2.6.5.3 Purge 2-147
2.7 PROGRAM MONITORING 2-149
2.7.1 Keypoint 2-149
2.7.2 Debug 2-149
2.7.2.1 Debug List 2-150
2.7.2.2 Debug Code (DC) 2-151
2.7.2.3 Debug Operation 2-152
2.7.2.4 Software Interface 2-154
2.7.2.4.1 Defined Interactions - Debug Enabled 2-154
2.7.2.4.2 Defined Interactions - Debug Not Enabled 2-155
2.7.2.4.2.1 Interactions with Debug 2-155
2.7.2.4.2.2 Enabling Debug 2-156

2.8 PROGRAM INTERRUPTIONS 2-157
2.8.1 Monitor Condition Register 2-166
2.8.1.1 Detected Uncorrectable Error (MCR48) 2-166
2.8.1.2 Not Assigned (MCR49) 2-166
2.8.1.3 Short Warning (MCR50) 2-167
2.8.1.4 Instruction Specification Error (MCR51) 2-167
2.8.1.5 Address Specification Error (MCR52) 2-167
2.8.1.6 CYBER 170 Exchange Request (MCR53) 2-168
2.8.1.7 Access Violation (MCR54) 2-168
2.8.1.8 Environment Specification Error (MCR55) 2-169
2.8.1.9 External Interrupt (MCR56) 2-170
2.8.1.10 Page Table Search Without Find (MCR57) 2-170
2.8.1.11 System Call (MCR58) 2-170
2.8.1.12 System Interval Timer (MCR59) 2-170
2.8.1.13 Invalid Segment/Ring Number Zero (MCR60) 2-171
2.8.1.14 Outward Call/Inward Return (MCR61) 2-171
2.8.1.15 Soft (or Corrected) Error (MCR62) 2-172
2.8.1.16 Trap Exception (MCR63) 2-172
2.8.2 Monitor Mask Register 2-173
2.8.3 User Condition Register 2-173
2.8.3.1 Privileged Instruction Fault (UCR48) 2-173
2.8.3.2 Unimplemented Instruction (UCR49) 2-173
2.8.3.3 Free Flag (UCR50) 2-174
2.8.3.4 Process Interval Timer (UCR51) 2-174
2.8.3.5 Inter-ring Pop (UCR52) 2-174
2.8.3.6 Critical Frame Flag (UCR53) 2-174
2.8.3.7 Not Assigned (UCR54) 2-174
2.8.3.8 Divide Fault (UCR55) 2-175
2.8.3.9 Debug (UCR56) 2-175
2.8.3.10 Arithmetic Overflow (UCR57) 2-175
2.8.3.11 Exponent Overflow (UCR58) 2-175
2.8.3.12 Exponent Underflow (UCR59) 2-175
2.8.3.13 Floating Point Loss of Significance (UCR60) 2-176
2.8.3.14 Floating Point Indefinite (UCR61) 2-176
2.8.3.15 Arithmetic Loss of Significance (UCR62) 2-176
2.8.3.16 Invalid BDP Data (UCR63) 2-176
2.8.4 User Mask Register 2-177
2.8.5 Exchange Operation and Interrupts 2-177
2.8.5.1 Job Process to Monitor Process Exchange 2-178
2.8.5.2 Monitor Process to Job Process Exchange 2-178
2.8.6 Trap Interrupt Operation 2-179
2.8.7 Multiple Interrupts 2-181
2.8.8 Enabling Interrupts 2-182
2.8.9 Interrupt Flowchart 2-183
2.8.10 Flags 2-186
2.9 BUFFERS 2-187
2.9.1 Map Buffer 2-187
2.9.2 Cache Buffer 2-187
2.9.3 Instruction Stack 2-188
2.9.4 Stack Frame Save Area (SFSA) Pushdown 2-188
2.9.4.1 General Operation of SFSA Pushdown 2-189

2.10 INTERFACES 2-190
2.10.1 Central Memory 2-190
2.10.1.1 Processor Central Memory Port Selection 2-190
2.10.2 Maintenance Access 2-190
2.11 PERFORMANCE MONITORING FACILITY (PMF) 2-191
2.11.1 PMF Initialization/Operation 2-193
2.11.2 PMF Status (Register 22 Byte 0) 2-193
2.11.3 PMF Control (Register 22 Bytes 1-7) 2-194
2.11.3.1 PMF Control Bits (Bytes 1-3) 2-194
2.11.3.2 PMF Control (Bytes 4-7) 2-194
2.11.4 PMF Counters (Register 22 Bytes 8-47) 2-195
2.11.5 Events and States 2-198
2.12 VECTOR INSTRUCTIONS 2-199
2.12.1 General Description 2-199
2.12.1.1 Format 2-199
2.12.1.2 Length (Number of Operations) 2-201
2.12.1.3 Broadcast 2-201
2.12.1.4 Interrupts 2-202
2.12.1.5 Results (Scalar/Vector) 2-202
2.12.1.6 Condition Register Bits 2-205
2.12.1.7 Overlap 2-206
2.12.1.8 Page Size 2-206
2.12.2 Integer Vectors - Arithmetic 2-207
2.12.3 Integer Vectors - Compare 2-207
2.12.4 Shift Vector Circular 2-208
2.12.5 Logical Vectors 2-209
2.12.6 Convert Vectors 2-209
2.12.7 Floating Point Vectors - Arithmetic 2-209
2.12.8 Floating Point Vector Summation 2-210
2.12.9 Merge Vector 2-210
2.12.10 Gather/Scatter Vectors 2-211
2.12.11 Floating Point Vectors - Triad 2-216
2.12.12 Floating Point Vector Dot Product 2-216
2.12.13 Gather/Scatter Vectors - Index List 2-217

3.0 VIRTUAL MEMORY MECHANISM 3-1
3.1 GENERAL DESCRIPTION 3-1
3.1.1 Level of Addresses 3-1
3.1.2 Address Components 3-1
3.1.2.1 Segments 3-2
3.1.2.2 Pages 3-2
3.1.3 Real Memory Address 3-3
3.2 PROCESS VIRTUAL ADDRESS 3-3
3.2.1 Format 3-3
3.2.1.1 Ring Number 3-4
3.2.1.2 Segment Number 3-4
3.2.1.3 Byte Number 3-4

3.3 PROCESS SEGMENT TABLE 3-5
3.3.1 Segment Descriptors 3-5
3.3.1.1 Control Fields 3-6
3.3.1.2 Access Validation Fields 3-6
3.3.1.3 Active Segment Identifier 3-6
3.3.1.4 Conversion to System Virtual Address 3-6
3.4 SYSTEM VIRTUAL ADDRESS 3-8
3.4.1 Active Segment Identifiers 3-8
3.4.2 Byte Number 3-9
3.4.2.1 Page Number 3-9
3.4.2.2 Page Size Mask Register 3-9
3.4.2.3 Page Offset 3-10
3.5 SYSTEM PAGE TABLE (SPT) 3-11
3.5.1 Page Descriptors 3-13
3.5.1.1 Control and Status Fields 3-13
3.5.1.2 Segment/Page Identifier (SPID) 3-13
3.5.1.3 Page Frame Address 3-13
3.5.2 Allocation of Page Descriptors 3-14
3.5.2.1 Location of a Page Descriptor in the Page Table 3-14
3.5.2.2 Search for Page Descriptor in the Page Table 3-18
3.5.2.3 Formation of the Real Memory Address (RMA) 3-18
3.6 ACCESS PROTECTION 3-19
3.6.1 Access Control Fields 3-19
3.6.2 Ring Hierarchy 3-19
3.6.2.1 Execute Ring Bracket 3-20
3.6.2.2 Read and Write Limits 3-20
3.6.2.3 Call Ring Limit 3-20
3.6.3 Key/Lock Facility 3-21
3.6.3.1 Format of Key/Lock Fields 3-21
3.6.3.2 Access Validations 3-22
3.6.3.3 Software Conventions 3-22

4.0 CENTRAL MEMORY 4-1
4.1 GENERAL 4-1
4.2 STANDARD MEMORY PORT INTERFACE 4-2
4.2.1 Interface Lines 4-2
4.2.2 Memory Functions, Responses, and Operations 4-5
4.2.2.1 Memory Functions 4-5
4.2.2.2 Memory Responses 4-5
4.2.2.3 Memory Operations 4-6
4.2.2.3.1 Read 4-6
4.2.2.3.2 Write 4-6
4.2.2.3.3 Read and Set Lock; Read and Clear Lock; Exchange 4-6
4.2.2.3.4 Read Free Running Counter 4-7
4.2.2.3.5 Refresh Counter Resync 4-7
4.2.2.3.6 Interrupt 4-7
4.2.2.4 Function and Response Code Interrelationships 4-8

4.3 MEMORY PERFORMANCE REQUIREMENTS 4-10
4.3.1 Ports 4-10
4.3.2 Distribution 4-10
4.3.3 Access Time 4-10
4.3.4 Bank Cycle Time 4-10
4.4 RAM FEATURES (SEE ALSO SECTION 8) 4-10
4.4.1 Parity 4-10
4.4.2 Single Error Correction/Double Error Detection - SECDED 4-10
4.4.3 Noninterleaved Mode 4-10
4.4.4 Memory Configuration Switches 4-11
4.4.4.1 OI Bit 23 Clear (Three-position switches) 4-11
4.4.4.2 OI Bit 23 Set (Two-position switches) 4-13
4.5 MAINTENANCE REGISTERS 4-14
4.5.1 Maintenance Registers Accessible by the Maintenance Access 4-15
4.5.2 Maintenance Registers Accessible by Memory Ports 4-15
4.5.2.1 Free Running Counter 4-15
4.6 BOUNDS REGISTER 4-16

5.0 INPUT/OUTPUT UNIT 5-1
5.1 GENERAL 5-1
5.2 PERIPHERAL PROCESSOR 5-2
5.2.1 Organization 5-2
5.2.1.1 Memory 5-2
5.2.1.2 Arithmetic Register 5-2
5.2.1.3 Arithmetic Logic Unit 5-3
5.2.1.4 Address registers 5-3
5.2.2 Instruction Set 5-4
5.2.2.1 Instruction Formats 5-4
5.2.2.2 Address Modes 5-5
5.2.2.2.1 No-Address Mode 5-5
5.2.2.2.2 Constant Mode 5-5
5.2.2.2.3 Direct Mode 5-6
5.2.2.2.4 Indirect Mode 5-6
5.2.2.2.5 Memory Mode 5-6
5.2.2.3 Nomenclature 5-7
5.2.2.4 General Instruction Notes 5-8
5.2.2.4.1 Short Word (12-Bit) Stores 5-8
5.2.2.4.2 Usage of PP Location 0 During Instruction Execution 5-8
5.2.2.4.3 Unused Bits 5-8
5.2.2.4.4 Compass Mnemonic 5-8
5.2.2.4.5 I0 and I4 PPs 5-8
5.2.2.4.6 I0 Instruction Usage Restriction 5-8
5.2.2.4.7 I0 Interrupt Processor Instruction 5-8
5.2.2.5 Load and Store 5-9
5.2.2.6 Arithmetic 5-11
5.2.2.7 Logical 5-13
5.2.2.8 Replace 5-17
5.2.2.9 Branch 5-21

5.2.2.10 Central Memory Access....5-22
5.2.2.10.1 R Register....5-22
5.2.2.10.2 Relocation Flag....5-22
5.2.2.11 Input/Output....5-36
5.2.2.12 Other....5-44
5.3 I/O CHANNELS....5-47
5.3.1 Internal Interface....5-47
5.3.1.1 Active bit....5-47
5.3.1.2 Full bit....5-47
5.3.1.3 Flag bit....5-48
5.3.1.4 Channel Error Flag....5-48
5.3.2 Real Time Clock....5-48
5.3.3 Two Port Multiplexer....5-49
5.3.3.1 General Description....5-49
5.3.3.2 Interface Definitions....5-50
5.3.3.2.1 Channel 15B to PPs....5-50
5.3.3.2.2 RS-232 Interface....5-50
5.3.3.2.3 RS-366A Interface....5-51
5.3.3.3 Characteristics....5-53
5.3.3.3.1 PP to Two Port Mux Function Codes....5-53
5.3.3.3.2 External Device to Two Port Mux Functions....5-67
5.3.3.3.3 Auto Answer....5-67
5.3.3.3.4 Remote Power Control....5-68
5.3.3.3.5 Remote Deadstart....5-71
5.3.3.4 Performance....5-73
5.3.3.4.1 Function Response Times....5-73
5.3.3.4.2 Data Transfer Rates....5-74
5.3.3.4.3 Calendar Clock Accuracy....5-75
5.3.3.5 Programming Considerations....5-75
5.3.3.5.1 RS-232 Interfaces....5-75
5.3.3.5.2 RS-366A Interface (Auto Dial-Out)....5-78
5.3.3.5.3 Calendar Clock....5-79
5.3.3.5.4 Loop Back....5-80
5.3.4 Maintenance Channel....5-81
5.3.5 External Interface....5-81
5.3.5.1 General....5-81
5.3.5.1.1 CYBER 170 External Interface....5-81
5.3.5.1.2 CYBER 180 External Interface....5-81
5.3.5.2 CYBER 170 and CYBER 180 Channel Control Signals....5-82
5.3.5.2.1 Active....5-82
5.3.5.2.2 Inactive....5-82
5.3.5.2.3 Full....5-82
5.3.5.2.4 Empty....5-82
5.3.5.2.5 Function....5-82
5.3.5.2.6 Master Clear....5-82
5.3.5.2.7 Error (CYBER 180 Channel Only)....5-82
5.3.5.2.8 10 MHz Clock (CYBER 170 Channel Only)....5-82
5.3.5.2.9 1 MHz Clock (CYBER 170 Channel Only)....5-82

5.3.5.3 (Section intentionally left blank) 5-82
5.3.5.4 Data signals 5-83
5.3.5.5 PP and Channel Interaction 5-83
5.3.5.5.1 Active Bit 5-83
5.3.5.5.2 Full Bit 5-83
5.3.5.5.3 Function Instructions 5-83
5.3.5.6 Transmission Characteristics 5-86
5.3.5.6.1 CYBER 170 Channel 5-86
5.3.5.6.2 CYBER 180 Channel 5-86
5.3.5.6.2.1 Signal 5-86
5.3.5.6.2.2 Cable 5-86
5.3.6 Data Transmission Errors 5-87
5.3.6.1 Data-In Transmission 5-87
5.3.6.2 Data-Out Transmissions 5-87
5.4 CACHE INVALIDATION 5-88
5.4.1 Central Write from d to (A) 5-88
5.4.2 Central Write (d) Words from m to (A) 5-88
5.5 INITIALIZATION 5-88
5.6 MAINTENANCE REGISTERS 5-89
5.6.1 OS Bounds (OSB) 5-90
5.7 RAM FEATURES 5-91
5.7.1 Error Detection 5-91
5.7.1.1 PP 5-91
5.7.1.2 I/O Channels 5-91
5.7.1.3 Central Memory Access 5-91
5.7.2 Error Recovery 5-91
5.8 INTERFACES TO OTHER SYSTEM ELEMENTS 5-92
5.8.1 Memory/IOU 5-92
5.8.1.1 Signals 5-92
5.8.1.1.1 Mark Lines 5-92
5.8.1.2 Functions 5-92
5.8.2 CPU/IOU 5-92
5.8.2.1 S2 Signals 5-93
5.8.2.1.1 Address 5-93
5.8.2.1.2 Buss 5-93
5.8.2.1.3 Exchange Code 5-93
5.8.2.1.4 Exchange Accept 5-93
5.8.2.1.5 Busy 5-93
5.8.2.2 S3 Signals 5-94
5.8.2.2.1 Address 5-94
5.8.2.2.2 Exchange Code 5-94
5.8.2.2.3 Invalidate 5-94
5.8.2.2.4 Exchange Accept 5-94
5.8.2.3 General Signals 5-94
5.8.2.3.1 Summary Status 5-94
5.9 PERFORMANCE MONITORING 5-95
5.9.1 Test Points Provided for Performance Monitoring 5-95
5.9.1.1 Channel Activity 5-95
5.9.1.2 PP Program Activity 5-95
5.9.1.3 PP to Central Memory Activity 5-95

6.0 **MAINTENANCE CHANNEL** 6-1
6.1 NOS, NOS/BE SUPPORT 6-4
6.2 MAINTENANCE PROCESSOR FEATURES 6-5
6.2.1 Stop/Start Capabilities 6-5
6.2.2 Exchange 6-7
6.2.3 Read/Write Functions 6-9
6.3 OPERATIONS 6-10
6.3.1 Initialization 6-10
6.3.2 Monitor and Report Errors 6-10
6.3.3 Fault Tolerance 6-10
6.3.4 Diagnostics 6-10
6.3.5 Fault Injection 6-10
6.4 RAM 6-10

7.0 **CYBER 170 STATE** 7-1
7.1 CYBER 180 OPERATING SYSTEM 7-1
7.2 CYBER 170 STATE MEMORY 7-2
7.2.1 Word Format 7-2
7.2.2 RAC, FLC, RAE and FLE 7-2
7.2.3 C170 P Register 7-2
7.2.4 CYBER 170 Memory Facilities 7-3
7.2.4.1 C170 Central Memory (CM) 7-3
7.2.4.2 Extended Memory 7-4
7.2.4.2.1 Extended Core Storage (ECS) 7-5
7.2.4.2.2 Extended Semiconductor Memory (ESM) 7-5
7.2.4.2.3 Unified Extended Memory (ECS Mode) 7-6
7.2.4.2.4 Unified Extended Memory (ESM Mode) 7-6
7.2.5 CYBER 170 Memory Image Segment 7-7
7.2.5.1 P Ring/Segment Number 7-7
7.2.5.2 Page Table Search Without Find (Page Fault) 7-8
7.2.5.3 Address Spec Error, Invalid Segment, Access Violation 7-8
7.2.5.4 Cache Purge 7-8
7.2.5.5 Mapping 7-9
7.3 CENTRAL PROCESSOR INSTRUCTION SET 7-10
7.3.1 Compare/Move Instructions 7-10
7.3.2 Trap 180 Instruction 7-10
7.3.3 Direct Read/Write Central Memory 7-10
7.3.4 Block Copy Instructions 7-11
7.3.4.1 ECS 7-13
7.3.4.2 UEM (ECS Mode) 7-20
7.3.4.3 UEM (ESM Mode) 7-22
7.3.5 Direct Read/Write Extended Memory 7-23
7.3.6 Read Free Running Counter 7-25
7.3.7 Central Exchange Jump to (Bj)+K (C170 Monitor Flag Set)
Central Exchange Jump to MA (C170 Monitor Flag Clear) 7-25
7.4 STATE SWITCHING BETWEEN CYBER 180 & CYBER 170 (C180/C170) .. 7-26
7.4.1 VMID 7-26

7.4.2 CYBER 180 Monitor to CYBER 170 State Exchange 7-26
7.4.2.1 P Register 7-29
7.4.2.2 Stack Pointers 7-29
7.4.2.3 RAC, FLC, MA 7-29
7.4.2.4 Exit Mode 7-29
7.4.2.5 RAE, FLE 7-30
7.4.2.6 A0-A7 7-30
7.4.2.7 B1-B7 7-30
7.4.2.8 X0-X7 7-30
7.4.2.9 Control Flags 7-31
7.4.2.10 CYBER 180 Ring Numbers 7-31
7.4.3 CYBER 170 State to CYBER 180 Monitor Exchange 7-32
7.4.4 Call to a CYBER 170 State Procedure from a CYBER 180 Job 7-32
7.4.5 Trap from CYBER 170 State to CYBER 180 State A 7-34
7.4.6 Return to a CYBER 170 Process 7-34
7.5 CYBER 170 EXCHANGE 7-35
7.5.1 CYBER 170 Exchange Packages 7-35
7.5.2 CYBER 170 Exchange Jump 7-36
7.5.3 Undefined Fields 7-37
7.5.4 RAE, FLE 7-37
7.6 ERROR HANDLING IN CYBER 170 STATE 7-38
7.6.1 Program Errors which Cause CPU Halt 7-38
7.6.2 Hardware Errors 7-38
7.6.3 Error Exit - C173/C170 State of C180 7-38
7.6.4 Address Out of Range 7-47
7.6.5 Parcel Boundaries 7-50
7.7 CODE MODIFICATION IN CYBER 170 STATE 7-51
7.8 CEJ/MEJ 7-51
7.9 DEBUG 7-51
7.10 CYBER 170 BREAKPOINT 7-51
7.11 READ CYBER 170 P REGISTER 7-52
7.12 CYBER 170 PP EXCHANGE REQUESTS 7-52
7.12.1 Exchange Jump 7-54
7.12.2 Monitor Exchange Jump 7-54
7.12.3 Monitor Exchange Jump to MA 7-54
7.13 EXTENDED CORE STORAGE (ECS) COUPLER 7-55
7.13.1 Interface to Central Memory 7-55
7.13.2 Interface to CPU 7-55
7.13.3 Initiating or Terminating from CPU 7-56
7.13.4 Cache Purge 7-57
7.13.5 Maintenance Channel and Registers 7-57
7.14 DUAL PROCESSOR C170 OPERATION 7-57
7.14.1 Cache Purge on C170 Exchange 7-57
7.14.2 Concurrent C170 Monitor Interlock Flag 7-58
7.14.3 PP Generated Interrupts 7-59
7.14.4 PP Initiated Cache Invalidations 7-59

8.0 RELIABILITY, AVAILABILITY, MAINTAINABILITY (RAM) 8-1
8.1 DEFINITIONS 8-1
8.1.1 System States 8-1
8.1.1.1 Fully Operational 8-1
8.1.1.2 Fault-tolerant Operation 8-1
8.1.1.3 Degraded Operation 8-1
8.1.1.4 Down 8-1
8.1.2 Interrupt 8-1
8.1.3 Deferred Maintenance 8-1
8.2 MINIMUM RAM FEATURES 8-2
8.2.1 Maintenance Processor 8-2
8.2.2 SECDED 8-2
8.2.3 Parity Checking 8-2
8.2.4 Degradable Cache and Map 8-2
8.2.5 Fault Isolation 8-2
8.2.6 Reconfiguration and Degradation 8-3
8.2.7 Instruction Retry 8-3
8.2.8 Micro Step Mode 8-4
8.2.9 Time-out 8-4
8.2.10 Power Supplies 8-4
8.2.11 Packaging 8-4
8.2.12 Forced Errors 8-5
8.2.13 Programmable Clock Margins 8-6
8.2.14 Component Failure Rates 8-6
8.2.15 Remote Technical Assistance (RTA) 8-6
8.2.16 Hardware Redundancy 8-7
8.3 ENVIRONMENTAL FAILURES 8-7
8.4 PERFORMANCE MONITORING 8-7

9.0 DEDICATED FAULT TOLERANCE (DFT) 9-1
9.1 ELEMENT STATUS SUMMARY 9-1
9.1.1 System Error Summary Reporting 9-1
9.1.1.1 Long Warning 9-1
9.1.1.2 Short Warning 9-2
9.1.1.3 Uncorrected Error 9-3
9.1.1.4 Corrected Error 9-4
9.1.2 Element Error Status Reporting 9-4
9.1.2.1 Process Not Damaged (PND) 9-4
9.1.2.2 Halt 9-5
9.1.2.3 Operating State and Mode 9-7
9.1.2.4 Static Configuration 9-7
9.1.2.5 Dynamic Configuration 9-7
9.1.2.6 Element Identification 9-7
9.1.2.7 Processor Environment 9-7
9.2 DETAILED ERROR REPORTING 9-8

Appendix A: CP Instructions in Reference Number Sequence A-1
Appendix B: CP Instructions in Operation Code Sequence B-1
Appendix C: Edit Examples C-1
Appendix D: Interrupt Conditions D-1
Appendix E: PP Instructions E-1
Appendix F: PP Instruction Address Modes F-1
Appendix G: Debug Conditions G-1
Appendix H: Edit Flowcharts H-1
Appendix I: Exception Conditions - UTP I-1
Appendix J: Hardware/Software Interaction J-1

Index Index-1

List of Figures

2.2-1 Register Selectivity Correspondence 2-18
2.5-1 CYBER 180 Exchange Package (C180 Process) 2-108
2.5-2 Stack Frame Save Area 2-117
2.6-1 Call/Return/Pop, Post-execution Stack Frame States 2-131
2.8-1 Interrupt Flowchart 2-185
2.11-1 PMF Register Formats 2-192
2.11-2 PMF Input Selectors and Counters 2-196
2.12-1 Gather Instruction 2-213
2.12-2 Scatter Instruction 2-215
3.1-1 Address Component Hierarchy 3-2
3.3-1 Conversion of PVA to SVA 3-7
3.4-1 System Virtual Address 3-8
3.4-2 Formation of Page Number and Page Offset 3-10
3.5-1 Transformation of SVA to RMA 3-15
3.5-2 Eight-bit Page Table Length 3-16
3.5-3 Fourteen-bit Page Table Length 3-17
4.1-1 Memory System Elements 4-1
5.2-1 CYBER 170 Mode R-Register (I1, I2, I4 and I0) 5-24
5.2-2 CYBER 180 Mode R-Register (I0 only) 5-25
5.3-1 Data Output Sequence 5-84
5.3-2 Data Input Sequence 5-85
6.0-1 CYBER 180 Maintenance Architecture 6-4
7.2-1 CYBER 170 Memories 7-3
7.3-1 011, 012 Instructions 7-12
7.3-2 014, 015 Instructions 7-24
7.4-1 CYBER 170 State Exchange Package Mapping 7-27
7.4-2 C180 Stack Frame Save Area Containing C170 Environment 7-33
7.5-1 CYBER 170 Exchange Package 7-35
7.13-1 ECS Protocol 7-56
9.1-1 CPU Error Recovery with CPU Stop on Error 9-6

List of Tables

1.3-1 Interelement Connection Alternatives 1-3
1.5-1 Typical Serial Numbers 1-4
1.5-2 Systems 1-7, 1-8, 1-9
1.5-3 Central Memory Capacities 1-10
1.6-1 Central Memory Port Assignments 1-12
1.6-2 Purge Buffer Instruction Sub-Ops 1-14
2.1-1 P Exception Testing 2-6
2.1-2 Unused Bits 2-8
2.4-1 64-Bit Floating Point Representation 2-69

Appendix A: CP Instructions in Reference Number Sequence A-1
Appendix B: CP Instructions in Operation Code Sequence B-1
Appendix C: Edit Examples C-1
Appendix D: Interrupt Conditions D-1
Appendix E: PP Instructions E-1
Appendix F: PP Instruction Address Modes F-1
Appendix G: Debug Conditions G-1
Appendix H: Edit Flowcharts H-1
Appendix I: Exception Conditions - UTP I-1
Appendix J: Hardware/Software Interaction J-1

Index Index-1

List of Figures

2.2-1	Register Selectivity Correspondence	2-18
2.5-1	CYBER 180 Exchange Package (C180 Process)	2-108
2.5-2	Stack Frame Save Area	2-117
2.6-1	Call/Return/Pop, Post-execution Stack Frame States	2-131
2.8-1	Interrupt Flowchart	2-185
2.11-1	PMF Register Formats	2-192
2.11-2	PMF Input Selectors and Counters	2-196
2.12-1	Gather Instruction	2-213
2.12-2	Scatter Instruction	2-215
3.1-1	Address Component Hierarchy	3-2
3.3-1	Conversion of PVA to SVA	3-7
3.4-1	System Virtual Address	3-8
3.4-2	Formation of Page Number and Page Offset	3-10
3.5-1	Transformation of SVA to RMA	3-15
3.5-2	Eight-bit Page Table Length	3-16
3.5-3	Fourteen-bit Page Table Length	3-17
4.1-1	Memory System Elements	4-1
5.2-1	CYBER 170 Mode R-Register (I1, I2, I4 and I0)	5-24
5.2-2	CYBER 180 Mode R-Register (I0 only)	5-25
5.3-1	Data Output Sequence	5-84
5.3-2	Data Input Sequence	5-85
6.0-1	CYBER 180 Maintenance Architecture	6-4
7.2-1	CYBER 170 Memories	7-3
7.3-1	011, 012 Instructions	7-12
7.3-2	014, 015 Instructions	7-24
7.4-1	CYBER 170 State Exchange Package Mapping	7-27
7.4-2	C180 Stack Frame Save Area Containing C170 Environment	7-33
7.5-1	CYBER 170 Exchange Package	7-35
7.13-1	ECS Protocol	7-56
9.1-1	CPU Error Recovery with CPU Stop on Error	9-6

List of Tables

1.3-1	Interelement Connection Alternatives	1-3
1.5-1	Typical Serial Numbers	1-4
1.5-2	Systems	1-7, 1-8
1.5-3	Central Memory Capacities	1-9
1.6-1	Central Memory Port Assignments	1-11
1.6-2	Purge Buffer Instruction Sub-Ops	1-13
2.1-1	P Exception Testing	2-6
2.1-2	Unused Bits	2-8
2.4-1	64-Bit Floating Point Representation	2-69

2.4-2 Floating Point Compare Results....2-91
2.4-3 Floating Point Sum Results, UM Clear....2-92
2.4-4 Floating Point Sum Results, UM Set....2-93
2.4-5 Floating Point Difference Results, UM Clear....2-94
2.4-6 Floating Point Difference Results, UM Set....2-95
2.4-7 Floating Point Product Results, UM Clear....2-96
2.4-8 Floating Point Product Results, UM Set....2-97
2.4-9 Floating Point Quotient Results (scalar), UM Clear....2-98
2.4-10 Floating Point Quotient Results (scalar), UM Set....2-99
2.4-11 Floating Point Quotient Results (vector), UM Clear....2-100
2.4-12 Floating Point Quotient Results (vector), UM Set....2-101
2.5-1 Bit Positions of Processor State Registers... 2-102
2.5-2 Real Memory Addresses....2-104
2.6-1 Register Definitions for "Copy"....2-145
2.6-2 Register Access Privilege....2-146
2.8-1 Monitor Condition Register....2-158
2.8-2 User Condition Register....2-158
2.8-3 Condition Registers, Bit Grouping....2-165
2.11-1 Definition of PMF Counter Actions....2-197
2.12-1 Vector Instructions....2-200
2.12-2 Vector Instruction Input and Output Fields....2-203, 2-204
3.5-1 Theoretical Maximum Central Memory (100% full Page Table)....3-12
3.5-2 Practical Maximum Central Memory (25% full Page Table)....3-12
4.1-1 Standard Memory Port....4-2
4.2-1 Function vs. Response Code for a Given Failure....4-8
4.2-2 Hardware Action Taken for Function vs. Failure....4-9
4.4-1 Memory Configuration Switches (three-position)....4-11
4.4-2 Memory Reconfigurations....4-12
4.4-3 Memory Configuration Switches (two-position)....4-13
4.5-1 Memory Register Access Privileges....4-14
5.3-1 CYBER 180 Channel Signal Definitions....5-86
5.6-1 Maintenance Registers....5-89
5.6-2 OS Bounds Register....5-90
6.0-1 Maintenance Channel Interface....6-1
6.0-2 Processor State Registers....6-2
6.0-3 Central Memory Maintenance Registers....6-2
6.0-4 IOU Maintenance Registers....6-3
7.2-1 C170 State Extended Memory....7-4
7.2-2 Extended Memory Flags....7-5
7.3-1 ECS Block Copy....7-14 – 7-19
7.3-2 UEM (ECS Mode) Block Copy....7-20
7.3-3 UEM (ESM Mode) Block Copy....7-22
7.4-1 Exchange Package Flags....7-28
7.6-1 Error Exits, C170 Monitor and Job Modes....7-39 – 7-46

1.0 INTRODUCTION

1.1 SCOPE

This General Design Specification is intended to define the common properties and characteristics of Central Processor models, Central Memory models, and Input/Output units which constitute major firmware/hardware components of the CDC CYBER 180 product line.* See table 1.5-2.

Included in this model-independent specification is the description of the Virtual Memory Mechanism commonly applicable to these major system components.

The use of *italic font* in this specification is explained in section 1.6, Model Differences: Machine States.

1.2 APPLICABLE DOCUMENTS

1.2.1 Control Documents

- CYBER 180 Architectural Objectives/Requirements, Doc. No. ARH1688
- CYBER 180 Configuration Notebook, Doc. No. ARH3386
- CYBER 180 II Assembler ERS, Doc. No. ARH3945
- CTI Interface Specification, Doc. No. ARH2948
- DFT/OS Interface Specification, Doc. No. ARH6853

1.2.2 Reference Documents

- CYBER 180 Clock System Specifications, Doc. Nos. 11896089/11896090
- CYBER 180 Performance Monitoring Facility Interface Specification
- CYBER 180 Processor/Memory Transmission Scheme Specification
- *CYBER 170/173 Engineering Specification, Doc. No. 19063000*
- CYBER 180 ECS Coupler Interface Requirements Specification, Doc. No. 11896624
- *CYBER 170 I/O Channel Transmission Circuit Specification, Doc. No. 19063800*

* In this internal specification, the term CYBER 180 also extends to components of the CYBER 900 product line.

1.3 CONFIGURATIONS

The architecture shall allow flexibility in the interconnection of the basic computer system elements. These elements shall consist of central processors, central memories and I/O units.

This specification addresses the ability to connect various system elements together, but does not define supported configurations.

For the purpose of this specification, the processors will be referred to as the five models P0, P1, P2, P3 or THETA processor. The central memory units will be referred to as the models M0, M1, M2, M3, or THETA memory. The Input/Output units will be referred to as the models I0, I1, I2, I4 or I4C. The systems will be referred to as S0, S1, S2, S3, or THETA.

A comprehensive list of system configurations is shown in table 1.5-2.

1.3.1 Interelement Transfer Paths

All data transfers between two central processors or between the I/O Unit and a central processor shall be via central memory. Transmission of data between central memories M2, M3 or THETA and the I2 or I4 Unit shall occur over logically compatible, 64-bit wide interfaces. The electrical interface and clocking are not necessarily compatible between all central memories and IOU models.

1.3.2 Interelement Clock

A detailed description of the clock system is included in the Clock System Specification listed in paragraph 1.2.2.

1.3.3 Interelement Connection Alternatives

Each processor shall provide one processor port (termed the local processor port) to access the central memory within its system. P2 shall also provide one processor port (termed the external processor port) to access a central memory in another system. Both processor ports on P2 shall be designed to interface to a standard memory port (4.1.7). The requirement for two processor ports on the P2 is implemented by providing both ports directly from the processor.

The I2 and I4 shall also be designed to interface to a standard memory port. An I4 attached to port 3 of a memory need not support the cache invalidation for C170 central memory writes (7.2.4) nor the C170 Exchange Request (7.12).

M3 and THETA memory ports 0 and 2 shall be appropriately designed (with regard to performance requirements) to interface the local processor port for P3 and THETA respectively. M3 and THETA memory ports 1 and 3 and all M2 memory ports shall be standard memory ports. Of these standard memory ports, port 3 of each memory shall not only be a standard memory port but also shall be capable of interfacing an element which is not within the same EMC boundary as the memory. The other standard memory ports may assume that the element attached is within the same EMC boundary as the memory.

An S0 or S1 system and elements therein are not required to directly interface any other elements on other systems.

These interconnection requirements are summarized in table 1.3-1. Note that any required resynchronization of clocks shall be performed by the element connecting to the standard memory port rather than by the memory.

MEMORY	PORT	P2	P3	THETA PROC.	I2	I4/I4C	CYBERPLUS & MAP V
M2	0 (std)	Yes (local)					
	1 (std)				Yes*		
	2 (std)	Yes (local)					
	3 (std)	Yes (ext'l)					Yes
M3	0		Yes (local)				
	1 (std)				Yes*	Yes*	
	2		Yes (local)				
	3 (std)					Yes	Yes
THETA MEMORY	0			Yes (local)			
	1 (std)					Yes*	
	2			Yes (local)			
	3 (std)					Yes	

*This IOU must be inside the same EMC boundry as the memory.

Table 1.3-1. Interelement Connection Alternatives

Interconnection alternatives between the I2, I4, CYBERPLUS or MAP V and central memories M2, M3 or THETA shall include options for one way electrical distances of one or two clock cycles of propagation delay.

There shall be a special processor termed the Maintenance Control Unit (MCU) which will form part of the I/O unit. Each of the central processors shall provide a Maintenance Channel (6.0) interface for the MCU. The MCU shall serve as the programmable maintenance facility for these processors.

1.4 GENERAL TIMING CONSIDERATIONS

Within each processor, instruction execution shall be "conceptually serialized." Although central memory and register references may occur out of order, (to whatever degree required by a processor's model-dependent implementation in the achievement of its cost/performance goals), the results from each of the associated instructions, as observed by the processor performing their execution, shall be the same as if such instructions were actually executed in a serialized fashion (i.e., each instruction's execution would be completed before the execution of any subsequent instructions would begin). The single exception to this concept shall occur in the case of self-modifying programs as stated in paragraph 2.1.2 of this specification.

Processor operations shall be further serialized, as observed by <u>other</u> processors, only to the extent that the function referred to as "serialization" is included within the execution of certain instructions as described in section 2.6 of this specification.

Program interruptions shall occur between the execution of instructions, and with timing precision relative to the cause of such interruptions, to the extent specified in section 2.8 of this specification.

1.5 SYSTEM ELEMENTS

Each element on CYBER 180 has two registers which identify that element uniquely. These registers are the Element Identifier (EID) and the Options Installed (OI). They are used during system initialization to determine the mainframe configuration. These two registers shall be constructed such that software shall not be able to change their contents.

Also discussed in this section are microcode naming conventions and system configuration information.

1.5.1 Element Identifiers (EID)

The EID has the following format:

32	40	48 63
Element No.	Model No.	Serial No.

The element number identifies the equipment as a processor, memory, IOU, etc. Element number assignments are as follows:

00 central processor
01 central memory
02 input/output unit

The hexidecimal model number further catagorizes elements. For example, a processor could be a P2, as distinguished from a P1 or a P3. See Table 1.5-2.

The serial number field is written in packed decimal notation (see 2.3.2.2). In this way, the console displays the literal EID.

SERIAL NO.	PACKED DECIMAL EQUIVALENT (48-63)			
0101	0000	0001	0000	0001
1019	0001	0000	0001	1001
0110	0000	0001	0001	0000
1489	0001	0100	1000	1001
1490	0001	0100	1001	0000

Table 1.5-1. Typical Serial Numbers

See table 1.5-2 for a list of supported system configurations.

1.5.2 Options Installed (OI)

The OI identifies the options installed on a given element. Examples are: channels and barrels on the IOU; cache or control store extensions on a processor; various memory increments on memory; various processor/memory/IOU configurations on the S1 system. The exact bit definitions of the OI are model-dependent and are specified in the appropriate engineering specification for each element. The range of central memory is shown in table 1.5-3.

1.5.3 Microcode Naming Convention

M rr s t xx

M = First character of the seven digit name to be M (microcode) to distinguish this name from the prior naming convention which used U.

rr = Hexidecimal numeric field (00-FF) to indicate the model number of the processor for which this microcode is intended.

s = Alphanumeric configuration descriptor to indicate on a model-dependent basis different microcodes with different capabilities.

t = Alpha descriptor to indicate specific microcode memory within the processor for which this microcode file is intended.

xx = Alphanumeric field to indicate the level of the microcode.

01 to 99 for released levels.
AA, AB...for specific "patches" to specific levels of microcode.

1.5.4 Configuration Guidelines

The specific microcode (rr and s) required for a specific processor is completely specified by the processor Model Number and Options Installed register. The required level (xx) of this microcode is not specified by anything within the processor.

The model number for a processor will change whenever hardware changes (as opposed to switch settings) are required within the processor such that the resulting processor is "different" from its predecessor model in a manner not achievable by the addition or deletion of options or FCOs.

Additional model numbers must be specified via an approved DAP.

The maintenance channel must be able to read the Options Installed register without any microcode support because the OI register helps specify which microcode will be loaded.

The setting of a bit in the OI may require a number of changes or other switch settings to provide the capability indicated. (For example, the setting of "C170 state available" on the second THETA processor might require recabling the cache invalidation buss to the second processor.)

When a capability is added to a processor which requires a different microcode, the impact may require including the ability to manually switch the associated bit in the OI register.

The addition of a capability which will be a permanent change to that processor merely requires that the OI be appropriately altered, not necessarily switchable. Any capability which must be switched periodically in or out will require an appropriately switchable bit in the OI register. No bits in the OI register will be software alterable.

Additions to the OI register must be specified via an approved DAP. This DAP must include a statement as to whether the bit requires a switch or may be "hard-wired".

Dynamic degradations of the processor as part of soft-fail procedures do not change the OI register.

The VMCL defines the environments in which a processor may execute. It does not define which microcode shall be loaded by the initialization software. For this reason, the VMCL does not need to be available via the maintenance channel until after the microcode has been loaded.

1.5.5 Systems Supported

The CYBER 180 systems supported by Control Data are itemized in table 1.5-2. The range of central memory capacities by CM model number is itemized in table 1.5-3. See section 1.6 for a list of systems supporting C170 state only, C180 state only, and dual C170/180 state operations.

```
The following annotation is used in this table:
     S3      = S3   Sn = System n
     S3CR    = S3 Cost Reduced            Pn = Central Processor n
     S3CR-   = S3 Cost Reduced Minus      Mn = Central Memory n
     S3CR--  = S3 Cost Reduced Minus Minus  In = Input/Output Unit n
     S3S     = S3S
```

		Element 00		Element 01	Element 02	
System		CPU 0 (model)	CPU 1 (model) (optional)*	CM (model)	IOU 0 (model)	IOU 1 (model) (optional)
S0-	930-A	P0- (58)		M0- (58)	I0- (58)	
S0-	930-B	P0- (5D)		M0- (5D)	I0- (5D)	
S0-	930-11	P0- (53)		M0- (53)	I0- (53)	
S0	930-31	P0 (52)		M0 (52)	I0 (52)	
S0e-	932-A	P0e- (5C)		M0e- (5C)	I0e- (5C)	
S0e-	932-8	P0e- (5F)		M0e- (5F)	I0e- (5F)	
S0e-	932-11	P0e- (55)		M0e- (55)	I0e- (55)	
S0e	932-31	P0e (54)		M0e (54)	I0e (54)	
S0e	932-32	P0e (54)	P0e* (54)	M0e (54)	I0e (54)	
S1 Slowed†		P1 Slowed(10)		M1 Slowed (10)	I1 Slowed(10)	
S1CR-	810	P1CR- (14)		M1CR- (14)	I1CR- (14)	
S1CR-	810A	P1CR- (14)		M1CR- (14)	I1CR- (14)	
S1CR-S	815S	P1CR-S (15)		M1CR-S (15)	I1CR-S (15)	
S1-	815	P1- (11)		M1- (11)	I1- (11)	
S1	825	P1 (12)		M1 (12)	I1 (12)	
S1CRS	825S	P1CRS (16)		M1CRS (16)	I1CRS (16)	
S1CR	830	P1CR (13)	P1CR (13)	M1CR (13)	I1CR (13)	
S1CR	830A	P1CR (13)	P1CR (13)	M1CR (13)	I1CR (13)	
S2	835	P2 (20)		M2 (20)	I2 (20)	

* 932-32 is a dual processor system only.
† All S1 Slowed systems, model number 10, have been upgraded to model 11 or 12.

Table 1.5-2. Systems (part 1 of 3)

The following annotation is used in this table:

S3 = S3 Sn = System n
S3CR = S3 Cost Reduced Pn = Central Processor n
S3CR- = S3 Cost Reduced Minus Mn = Central Memory n
S3CR-- = S3 Cost Reduced Minus Minus In = Input/Output Unit n
S3S = S3S I4AC = I4 (air-cooled)

System		Element 00		Element 01	Element 02	
System		CPU 0 (model)	CPU 1 (model) (optional)*	CM (model)	IOU 0 (model)	IOU 1 (model) (optional)
S3CR--	840	P3CR-- (34)		M3CR (31)	I2 (20)	
S3CR--	840	P3CR-- (34)		M3CR (31)	I4 (40)	
S3CR--	840A	P3CR-- (34)		M3CR (31)	I4 (40)	I4 (40)
S3S--	840S	P3S-- (37)		M3CR (31)	I4 (40)	
S3-	845	P3- (31)		M3 (30)	I2 (20)	
S3-	845	P3- (31)		M3 (30)	I4 (40)	
S3-	845	P3- (31)		M3CR (31)	I2 (20)	
S3-	845	P3- (31)		M3CR (31)	I4 (40)	
S3S-	845S	P3S- (35)		M3CR (31)	I4 (40)	
S3CR-	850	P3CR- (33)		M3CR (31)	I2 (20)	
S3CR-	850	P3CR- (33)		M3CR (31)	I4 (40)	
S3CR-	850A	P3CR- (33)		M3CR (31)	I4 (40)	I4 (40)
S3	855	P3 (30)		M3 (30)	I2 (20)	
S3	855	P3 (30)		M3 (30)	I4 (40)	
S3	855	P3 (30)	P3CR (32)	M3CR (31)	I2 (20)	
S3	855	P3 (30)	P3CR (32)	M3CR (31)	I4 (40)	
S3S	855S	P3S (36)		M3CR (31)	I4 (40)	
S3CR	860	P3CR (32)	P3CR (32)	M3CR (31)	I2 (20)	
S3CR	860	P3CR (32)	P3CR (32)	M3CR (31)	I4 (40)	
S3CR	860A	P3CR (32)		M3CR (31)	I4 (40)	I4 (40)
S3CR	870A	P3CR (32)	P3CR* (32)	M3CR (31)	I4 (40)	I4 (40)

* 870A is a dual processor system only.

Table 1.5-2. Systems (part 2 of 3)

The following annotation is used in this table:

S3	= S3	Sn	= System n
S3CR	= S3 Cost Reduced	Pn	= Central Processor n
S3CR-	= S3 Cost Reduced Minus	Mn	= Central Memory n
S3CR--	= S3 Cost Reduced Minus Minus	In	= Input/Output Unit n

		Element 00		Element 01	Element 02	
System		CPU 0 (model)	CPU 1 (model) (optional)*	CM (model)	IOU 0 (model)	IOU 1 (model) (optional)
PIP3-	960-11	PIP3- (3B)		PIM3 (34)	I4AC (40)	I4C (44)
PIP3-	960-11S	PIP3- (3B)		PIM3 (34)	I4S (42)	I4C (44)
PIP3-	962-11	PIP3- (3B)		PIM3 (34)	I4C (44)	I4C (44)
PIP3	960-31	PIP3 (3A)		PIM3 (34)	I4AC (40)	I4C (44)
PIP3	960-31S	PIP3 (3A)		PIM3 (34)	I4S (42)	I4C (44)
PIP3	960-32	PIP3 (3A)	PIP3* (3A)	PIM3 (34)	I4AC (40)	I4C (44)
PIP3	960-32S	PIP3 (3A)	PIP3* (3A)	PIM3 (34)	I4S (42)	I4C (44)
PIP3	962-31	PIP3 (3A)		PIM3 (34)	I4C (44)	I4C (44)
PIP3	962-32	PIP3 (3A)	PIP3* (3A)	PIM3 (34)	I4C (44)	I4C (44)
Theta	990	(40)	(40)	16K ECL (40)	I4 (40)	I4 (40)
Theta	990	(40)	(40)	64K CMOS (41)	I4 (40)	I4 (40)
Theta	990	(40)	(w/dp)† (41)	16K ECL (40)	I4 (40)	I4 (40)
Theta	990	(40)	(w/dp)† (41)	64K CMOS (41)	I4 (40)	I4 (40)
Theta	990	(w/dp)† (41)	(w/dp)† (41)	16K ECL (40)	I4 (40)	I4 (40)
Theta	990	(w/dp)† (41)	(w/dp)† (41)	64K CMOS (41)	I4 (40)	I4 (40)
Theta	990E	(w/dp)† (41)		64K CMOS (41)	I4 (40)	I4/I4C(40/44)
Theta	995E	(w/dp)† (41)	(w/dp)†* (41)	64K CMOS (41)	I4 (40)	I4/I4C(40/44)
Theta	990E	(w/dp)† (41)		256K CMOS (42)	I4 (40)	I4/I4C(40/44)
Theta	995E	(w/dp)† (41)	(w/dp)†* (41)	256K CMOS (42)	I4 (40)	I4/I4C(40/44)
Theta	992-31	(42)	(42)	256K CMOS (42)	I4C (44)	I4C (44)
Theta	992-32	(42)	* (42)	256K CMOS (42)	I4C (44)	I4C (44)
Theta	994-31	(44)	(44)	256K CMOS (42)	I4AC (40)	I4/I4C(40/44)
Theta	994-32	(44)	* (44)	256K CMOS (42)	I4AC (40)	I4/I4C(40/44)
Theta-E-	2000S-1xx	(wo/vect)‡ (46)	(wo/vect)‡ (46)	DRAM (46)	I4CE (46)	I4CE (46)
Theta-E	2000V-1xx	(w/vect) (48)	(w/vect) (48)	BIMOS (48)	I4CE (46)	I4CE (46)

* The 960-32, 960-32A, 962-32, 992-32, 994-32 and 995E are dual processor systems only.
† Theta CPU model 41 includes the high performance double precision floating point unit.
‡ Theta-E Minus CPU model 46 does not include vector instructions.

Table 1.5-2. Systems (part 3 of 3)

Table 1.5-3 states the capacities in megabytes (MB) of the Central Memories listed in Table 1.5-2.

Model	System		Mem. Chip	Mem. Range
(11)	M1-	815	64K	2-8 MB
(11)	M1-	815	256K	16-32 MB
(12)	M1	825	64K	2-8 MB
(12)	M1	825	256K	16-32 MB
(13)	M1CR	830	64K	2-16 MB
(13)	M1CR	830/830A	256K	16-64 MB
(14)	M1CR-	810	64K	2-16 MB
(14)	M1CR-	810	256K	16-64 MB
(14)	M1CR-	810A	256K	8-64 MB
(15)	M1CR-S	815S	64K	2-8 MB
(16)	M1CRS	825S	64K	2-8 MB
(20)	M2	835	16K	4-16 MB
(30)	M3	845/855	16K	4-16 MB
(31)	M3CR	840/840A/840S	256K DRAM	16-128 MB
(31)	M3CR	845/845S	" "	" MB
(31)	M3CR	850/850A	" "	" MB
(31)	M3CR	855/855S	" "	" MB
(31)	M3CR	860/860A/870A	" "	" MB
(34)	PIM3	960/962	1M DRAM	64-256 MB*
(40)	Theta	990	16K ECL	8-32 MB
(41)	Theta	990/990E/995E	64K CMOS	16-128 MB
(42)	Theta	990E/995E/992/994	256K CMOS	64-256 MB
(46)	Theta-E-	2000S	1M DRAM	128-512 MB
(48)	Theta-E	2000V	256K BIMOS	128-256 MB
(52)	M0	930-31	256K	8-64 MB
(52)	M0	930-31	1M	64-128 MB
(53)	M0-	930-11	256K	8-64 MB
(53)	M0-	930-11	1M	64-128 MB
(54)	M0e	932-31/-32	256K	8-64 MB
(54)	M0e	932-31/-32	1M	64-128 MB
(55)	M0e-	932-11	256K	8-64 MB
(55)	M0e-	932-11	1M	64-128 MB
(58)	M0-	930-A	256K	8-16 MB
(5C)	M0-	932-A	256K	8-16 MB
(5D)	M0-	930-B	256K	8-16 MB
(5F)	M0-	932-B	256K	8-16 MB

* 962-11 comes with a 32 MB capacity.

Table 1.5-3. Central Memory Capacities

1.6 MODEL DIFFERENCES

The following difference between models shall be noted.

Machine States

All of the CYBER 180 mainframes (with the exception of the 815S and 825S) support the C180 state machine, the definition of which constitutes the majority of this specification. Some mainframes also execute CYBER 170 code (or contain a C170 state machine). Chapter 7 describes the C170 state machine. Because references to the C170 state machine appear in other chapters as well, *all C170 specific material is printed in italics to aid the reader.* These two are the only supported machine states and are assigned identifiers as specified in 2.5.1.12.

C170 State

The following systems support C170 state operations:

S1 (810, 815, 825, 830) Note: 815S and 825S are C170 state only.
S2 (835)
S3 (840, 845, 850, 855, 860, 870)
PIP3 (960) [Note: the PIP3 model 962 is a C180 state system only]
Theta (990 and 990E)

Only the S1CR processor (model number 13) supports concurrent C170 execution in dual processor systems. (See 7.14.)

CMU

The S1 and S3 execute the C170 Compare Move Unit (CMU) instructions (Op codes 464 through 467) in the hardware. All other models having C170 State detect these as Unimplemented Instructions. (See 7.3.1.)

Performance Monitoring Facility (PMF)

Initially, PMF was required as an option on all C180 models. Subsequently this requirement was removed from all models except THETA (990, 990E), where it remains a standard feature. For THETA-E, the PMF is included neither in the standard system nor as an option; however, the PMF has been documented for THETA-E as a tool for development and manufacturing. Some early non-THETA models may still exist that were manufactured with the PMF option. (See 2.11.)

Bounds Register

The CYBER 180 architecture originally specified a 64-bit Bounds Register. This is the format found in all processors of the S1 (810-830), S2 (835), S3 (840-870), and THETA (990, 990E).

This 64-bit register has 4 bits reserved for the Bit Vector for Port Bounds, 16 bits reserved for the Upper Bounds, and 16 bits reserved for the Lower Bounds. Figure 1.6-1 illustrates the 64-bit Bounds Register format.

```
0   3 4                         31 32             47 48             63
+-----+----------------------------+-----------------+-----------------+
|     |        Not assigned        |  Upper Bounds   |  Lower Bounds   |
+-----+----------------------------+-----------------+-----------------+
   |
   L--Bit Vector for Port Bounds (bit 0 indicates port 0, etc.)
```

Figure 1.6-1. Bounds Register (64 bits)

Setting a bit in the Bit Vector for Bounds confirms the corresponding port in the following way. Writes are inhibited for all addresses greater than or equal to the Upper Bounds or less than the Lower Bounds. An Uncorrectable Error Response (table 4.2-1) is returned and an entry is placed in Uncorrected Error Log 1. A read operation is not tested for bounds.

The 16 bits of the Upper Bounds and Lower Bounds represent bits 36 through 51 of the Real Memory Address (RMA bits 34 and 35 are reserved). This provides a maximum bounds of 256 megabytes with the address bounds represented in modulo 4K bytes.

The S0 (930), PIP3 (960, 962), and all subsequent processors use the Bounds Register format defined in section 4.6.

Debug on the Branch on Condition Register

The debug test on the Branch on Condition Register instruction has been temporarily removed from the 840, 845S and 855S processors. It shall be reinstated as soon as microcode space permits. (See 2.7.2.2, Appendix D [D-10], and Appendix G [G-5]).

Memory Configuration Switches

Memory configuration switches as specified in paragraph 4.4.4 are implemented in the following central memory models only:

M1 Models 10, 11, 12, 13, 14
M3 Models 30, 31
M2 Model 20 THETA Model 40

No other models contain these switches.

Central Memory Port Assignments

Model	Port Number	Type	Assignments
M0	0	Nonstandard	IO
	1	Nonstandard	IO
	2	Nonstandard	Processor
	3	Nonstandard	Processor
M1	0	Nonstandard	Primary Processor - P1
	1	Nonstandard	I1(Interface A)
	2	Nonstandard	Optional Processor - P1
	3	Nonstandard	I1(Interface B)
M2	0	Nonstandard	Primary Processor - P2
	1	Standard	IOU
	2	Nonstandard	Optional Processor - P2
	3	Standard	*(21 address bits)
M3	0	Nonstandard	Primary Processor - P3
	1	Standard	IOU
	2	Nonstandard	Optional Processor - P3
	3	Standard	*(27 address bits)
THETA	0	Nonstandard	Primary Processor - THETA
	1	Standard	IOU
	2	Nonstandard	Optional Processor - THETA
	3	Standard	*(27 address bits)

* External port available for MAP-V, CYBERPLUS or other future hardware meeting the Standard Memory Interface requirements.

Table 1.6-1. Central Memory Port Assignments

Standard Memory Port Interface

The standard port shall be capable of accepting memory requests as follows:

M2 - one request every 56 ns
M3 - one request every 64 ns
PIP3 - one request every 80 ns
THETA - one request every 64 ns

Page Table Length Register

There are two versions of the PTL Register as described in 2.5.1.4. All C180 models have the 8-bit PTL except as noted below, all of which have the 14-bit PTL:

S0-E (932)
PIP3 (960, 962)
THETA-E (997)
and all subsequent systems

Monitor Condition Register, Bit 48

MCR48 is implemented either as DUE Software Flag or as Detected Uncorrectable Error, as described in 2.8.1.1 and shown in Table 2.8-1.

CPU models 10-14, 20, 30-37, 3A, 3B, 40-41 and 50-53 (see Table 1.5-2) implement MCR48 as Detected Uncorrectable Error. See 2.8.1.1.2. Within this group, CPU models 3A, 3B, 40 and 41 have implemented Stop on Error along with MCR48. See 2.8, Group 0. On these four models, MCR48, if set on Stop on Error, indicates that an uncorrected, nonretryable error condition has been detected. On processor restart, the Maintenance Processor restores MCR48 to the definition specified in 2.8.1.1.2.

All other CPU models implement MCR48 as DUE Software Flag. See 2.8.1.1.1.

Vector Instructions

Vector instructions are implemented only on Theta systems. System models 990 and 995 implement only the standard vector instructions as listed in table 2.12-1. System model 997 implements both the standard and the extended vector instructions as listed in table 2.12-1.

Global Key and SPI Identifier

The following systems implemented a global key lock mechanism defined in this document prior to DAP ARH5792:

S1 (810, 815, 825, 830)
S2 (835)
S3 (840, 845, 850, 855, 860, 870)
PIP3 (960, 962)
Theta (990 and 990E)

DAP ARH5792 specified a simplified key lock structure as defined in 3.6.3.1. This redefines the global key field in the P register as the SPI identifier. The models listed above support the SPI field as a subset of the original key lock definition.

S0 systems (930) support neither the global key lock mechanism nor the SPI identifier field.

The S0-E (932), Theta-E (997), Vanguard, and all subsequent models shall support the SPI field as defined in 2.5.2.29.

Instruction Stack Purge

Several of the sub-ops (specifically k=4 through 6, and k=C through E) of the Purge Buffer instruction (Op. 05) have been changed during the course of the C180 product line development. While these were changed in a manner to preserve compatibility at the time of the change, there are some performance considerations for any situation utilizing the k=4, purge instruction stack sub-op. Additionally, any future changes in this instruction require careful consideration of these differences.

<table>
<tr><th>k Sub-op</th><th>S0</th><th>S1</th><th>S2</th><th>S3</th><th>THETA</th><th>THETA-E</th></tr>
<tr><td>0-2</td><td colspan="6">Per Specification</td></tr>
<tr><td>3</td><td colspan="6">Per Specification</td></tr>
<tr><td>4</td><td colspan="5" rowspan="1">As if k=7</td><td>Purge Inst Stack</td></tr>
<tr><td>5-6</td><td colspan="4">As if k=7</td><td colspan="2">No-op</td></tr>
<tr><td>7</td><td colspan="6">Per Specification</td></tr>
<tr><td>8-B</td><td colspan="6">Per Specification</td></tr>
<tr><td>C-E</td><td colspan="4">As if k=F*</td><td colspan="2">No-op</td></tr>
<tr><td>F</td><td colspan="6">Per Specification</td></tr>
</table>

*Local Privilege is required

Table 1.6-2. Purge Buffer Instruction Sub-Ops

Ring Number Zero

The 932 and the Theta-E do not test for ring number zero at any point. All other models perform the ring number zero test as specified in paragraphs 2.2.1.6, 2.2.1.7, 2.6.1.4, 2.6.1.5, 2.8.1.13, and 3.2.1.1. No future C180 models will perform the ring number zero test.

SFSA Pushdown

Theta-E is the only C180 processor to contain an implementation of the Stack Frame Save Area (SFSA) Pushdown as described in section 2.9.4. The SFSA Pushdown in Theta-E stores the P register, the VMID and the SFSA Descriptor.

Short Warning

On the Theta-E, the Short Warning bit (MCR50) is permanently forced to zero. MCR50, therefore, cannot be set implicitly by an environmental failure or explicitly by the Branch on Condition Register instruction or by exchanging to a process having MCR50 set in the exchange package. Environmental failures on Theta-E are managed by the Service processor.

All C180 models other than Theta-E have implemented the Short Warning bit as per the description in section 2.8 and elsewhere in this specification.

2.0 PROCESSOR

Central processor models P1-P3 and Theta shall provide the means for reading and translating each of the instruction codes contained in the instruction repertoire, as well as performing the corresponding execution of these instructions as defined by the descriptions contained in this specification.

In order to accomplish instruction fetch and execution, each processor shall additionally provide the means for referencing central memory. Central memory references shall be performed in virtual mode, which shall include the address translation and protection facilities as described in Sections 3.0 through 3.6 of this specification.

2.1 GENERAL DESCRIPTION

For the purposes of this specification the operation codes from the instruction repertoire shall be divided into four groups of instructions referred to as the General Instructions, the Business Data Processing Instructions, the Floating Point Instructions, and the System Instructions. In addition to central memory, addressed in virtual mode, the execution of the instructions within the first three of these instruction groups, namely the General, BDP, and Flt. Pt. Instructions, shall require the means to reference general containers referred to as the P Register, the A Registers, and the X Registers. Also, the means for detecting and indicating exceptional conditions, which may occur in the course of executing these instructions, shall be provided in accordance with the appropriate instruction descriptions contained in this specification.

The fourth group, namely the System Instructions, shall additionally require the means to reference special containers referred to as the Processor State Registers, Process State Registers, and Memory Maintenance Registers in accordance with the appropriate descriptions contained within sections 2.5, 2.6, and 4.5 of this specification, respectively.

2.1.1 General Registers

The means for referencing a total of 33 General Registers shall be provided.

2.1.1.1 P Register

The Program Address Register, referred to simply as the P Register, shall consist of 64 bits, numbered from left to right, beginning with bit position 00. Conceptually, the P Register shall contain the Process Virtual Address, PVA, of an instruction in central memory during the time it is <u>read</u>, <u>interpreted</u>, and <u>executed</u> by the processor. Similarly, the P Register shall contain a Key to central memory during each instruction's execution. The contents of the P Register shall be formatted as follows: where the RN (Ring Number), SEG (Segment) and BN (Byte Number) fields are individually described within section 3.2 of this specification, and the Key field is described within paragraph 3.6.3, and SPI in section 2.5.2.29 of this specification.

Bits 0, 1, 8 and 9 of the P register shall be set to zeros by the software responsible for generation of exchange packages. Any software writing the P register via the maintenance access is responsible to write zeros in bits 0, 1, 8 and 9. Failure to provide zeros either in the exchange package or via the maintenance access write or via a copy instruction shall result in undefined operation.

0	2	8	10	16	20	32 — 63
0 0	SPI	0 0	Key	RN	SEG	BN
2	6	2	6	4	12	32
				<------------	PVA	------------>

The processor shall provide zeros for bits 0, 1, 8, and 9 when the P register is read or stored into the exchange package or into the Stack Frame Save Area (SFSA). Bits 32 and 63 are copied from the input on loads of P and copied into output on stores and reads of P. See table 2.1-2.

2.1.1.2 A Registers

The sixteen A Registers, referred to as the A0 Register through the AF Register (using hexadecimal notation), shall consist of 48 bits each, identical in format to the rightmost 48 bits of the P Register as just previously described.

Note: Although these address registers are intended for general use in explicitly supplying such PVA's as may be required for branch (jump) and operand references to central memory, an aggregate of five A Registers, (namely, A0 through A4), shall be implicitly utilized during CALL instruction executions as described in section 2.6 of this specification.

2.1.1.3 X Registers

The sixteen X Registers, referred to as the X0 Register through the XF Register (using hexadecimal notation), shall consist of 64 bits each with their bit positions numbered from left to right, beginning with bit position 00, as follows:

```
|00                             31|32                             63|
|---------------------------------|---------------------------------|
|    X Register Left (32 bits)    |   X Register Right (32 bits)    |
|<---------------------- 64-Bit X Register ----------------------->|
```

The 64-bit contents of an X Register may be treated as a logical quantity, a signed binary integer, or a signed floating point number. Bit string, byte string, 32-bit halfword (right-justified in bit positions 32 through 63), and 64-bit word operations shall be provided for the contents of the X Registers.

Store operations to Xk left (XkL) shall not alter Xk right (XkR) and store operations to Xk/X0/X1 right shall not alter Xk/X0/X1 left.

Note: Although these operand registers are intended for general use in explicitly supplying such operands as may be required for accomplishing the execution of a majority of instructions, the first two X Registers, (namely, X0 and X1), shall be implicitly utilized during certain instructions which require additional input arguments or execution results. In these cases, Register X0 Right shall normally be used to supply additional input parameters to instruction execution and Register X1 Right shall be utilized to receive additional results from instruction execution. Whenver applicable, the instruction descriptions contained in this specification will fully define all register utilizations which shall be implicit in nature, including those cases in which the contents of Register X0 shall be interpreted as consisting, partially or entirely, of zeros.

2.1.2 Programming Restrictions

Programmed modification of the instructions comprising a stored program in central memory may lead to undefined results for instructions in the instruction stack (buffer). On exchange operations, the instruction stack is cleared as described in paragraph 2.8.5.

2.1.3 Instructions

Instructions shall be 16 bits or 32 bits in length, according to one of the four formats described in the following subparagraphs.

2.1.3.1 Formats jkiD and SjkiD

Operation Code	j	k	i	D
8	4	4	4	12

Operation Code	S	j	k	i	D
5	3	4	4	4	12

Nonvector instructions: the j, k and i fields shall provide register designations, the D field shall provide either a signed shift count, a positive displacement or a bit-string descriptor, and the S field shall provide a suboperation code.

Within the BDP instruction group, one or two descriptors shall be appended to instruction format jkiD. (See section 2.3.)

Vector instructions: the j, k and i fields shall provide register designations (A registers containing the starting address of a vector or X registers containing an immediate operand) and the D field containing both the length of the vector operation and the broadcast selection.

2.1.3.2 Format jk

Operation Code	j	k
8	4	4

For this 16-bit instruction format, the j field shall provide a register designation, a suboperation code, or an immediate operand value and the k field shall provide a register designation or an immediate operand value. Within the BDP instruction group, two descriptors shall be appended to this instruction format. (See section 2.3.)

2.1.3.3 Format jkQ

Operation Code	j	k	Q
8	4	4	16

For this 32-bit instruction format, the j and k fields shall provide register designations, suboperation codes, or an immediate operand value. The 16-bit Q field shall provide a signed displacement or an immediate operand value.

2.1.3.4 P Address Access

The P address may produce any of the four exception conditions described below. The P exception testing and handling on exchange operations is described in paragraph 2.8.8. The following paragraphs and table 2.1-1 describe the exception testing for sequential instruction fetches and for C180 branch instruction and Trap operations:

Invalid Segment

Four operations (Return, Inter-segment Branch, Call Indirect, Trap) are capable of generating a new segment number for P. When any of these four attempts to load P with an invalid segment number, an Invalid Segment shall be recorded; the instruction or operation shall be inhibited and the corresponding program interruption shall occur.

Access Violation

For the purpose of establishing central memory access validation, the reading of every instruction shall be an Execute type access. When specifically included within an instruction's description, the appropriate central memory access, performed for the purpose of fetching the instruction to be subsequently executed, shall be execute validated. Execute type accesses shall use the ring number contained in the P Register for access validation. The access validation procedure, which requires the classification of central memory accesses into read, write, execute, and call types, is described in section 3.6 of this specification.

Four operations (Return, Inter-segment Branch, Call Indirect, Trap) are capable of generating a P with an access violation. When any of these four attempts to do this, an Access Violation shall be recorded, the instruction or operation shall be inhibited and the corresponding program interruptions shall occur.

Address Specification Error

With the exception of the 9X instructions, when any of the instructions or operations listed in table 2.1-1 attempts to write a byte number having bit 32=1 into P, an Address Specification Error shall be recorded, this instruction or operation shall be inhibited and the corresponding program interruption shall occur. For all 9X instructions, the processors shall be allowed to either inhibit or complete execution as necessary to maximize performance. If no clear performance or cost advantage is present for either solution, the processor shall inhibit the 9X instructions for bit 32=1 Address Specification Errors.

Instruction accesses shall be confined to byte addresses which are 0, modulo 2. (The 32-bit instructions are not restricted to be in a single central memory word.) Four operations (Return, Inter-segment Branch, Call Indirect, Trap) are capable of generating a P with bit 63=1. When any of these four attempts to do this, an Address Specification Error shall be recorded, the instruction or operation shall be inhibited and the corresponding program interruption shall occur.

When any of the instructions or operations listed in table 2.1-1 writes an address having bit 32=0 into P while another parcel or BDP descriptor associated with the new instruction sets bit 32, the branch operation shall complete and the Address Specification Error shall be recorded as part of the attempted fetch of the new instruction.

Page Table Search Without Find

When any of the instructions or operations listed in table 2.1-1 writes an address into P such that the corresponding page is not in memory, the processors shall be allowed to either inhibit or complete the branch operation as necessary to maximize performance. The entries in table 2.1-1 represent the current implementation.

When any of the instruction or operations listed in table 2.1-1 writes an address into P such that the first parcel of the new instruction is in memory but another parcel or BDP descriptor associated with the new instruction is not, the processor shall complete the branch operation and the Page Table Search Without Find shall be recorded as part of the attempted fetch of the new instruction.

KEY:

INHIBIT COMPLETE

(SEE NOTE)

INSTRUCTION	INVALID SEGMENT/ RING NUMBER ZERO	ACCESS VIOLATION	ADDRESS SPECIFICATION ERROR, BIT 63=1	ADDRESS SPECIFICATION ERROR, BIT 32=1	ADDRESS SPECIFICATION ERROR, BIT 32=0*	PAGE TABLE SEARCH WO/FIND (PARCEL 0)	PAGE TABLE SEARCH WO/FIND (PARCEL >0)
	MCR60	MCR54	MCR52			MCR57	
04 RETURN	ALL –	ALL –	ALL –	ALL –	– ALL	P1,P2 P3,θ	– ALL
2E BRANCH RELATIVE				ALL –	– ALL	P1,P2 P3,θ	– ALL
2F INTER-SEGMENT BRANCH	ALL –	ALL –	ALL –	ALL –	– ALL	P1,P2 P3,θ	– ALL
90-9E CONDITIONAL BRANCH				P1 P2,P3,θ	– ALL	P1,P2 P3,θ	– ALL
9F BRANCH ON CONDITION REG.				P1 P2,P3,θ	– ALL	P1,P2 P3,θ	– ALL
B0 CALL RELATIVE				ALL –	– ALL	P1,P2 P3,θ	– ALL
B4 COMPARE SWAP				ALL –	– ALL	P1,P2 P3,θ	– ALL
B5 CALL INDIRECT	ALL –	ALL –	ALL –	ALL –	– ALL	P1,P2 P3,θ	– ALL
– TRAP	ALL –	ALL –	ALL –	ALL –	– ALL	P1,P2 P3,θ	– ALL

INHIBIT – P IN EXCHANGE PACKAGE OR STACK FRAME SAVE AREA WILL POINT AT THE BRANCH INSTRUCTION.

COMPLETE – P WILL POINT AT THE DESTINATION INSTRUCTION.

*Final P has bit 32=0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32=1.

Table 2.1-1. P Exception Testing

2.1.3.5 Unused Bits

a. Instructions

When one or more bits from an instruction are unused, i.e., their value(s) and associated function(s) are not specified within the instruction description, the execution of these instructions shall not be affected by the values of these bits. However, it is recommended that such bits are equal to zeros.

b. Processor Registers and Process Registers

Table 2.1-2 summarizes definitions for the unused bits in processor and process registers. Note that this table describes loading and storing operations for:

Copy Instructions
Maintenance Access Read/Writes
Exchange Operations
Stack Frame Save Area References

Notes for Table 2.1-2

a. UNUSED/ZERO - These bits are unused (ignored by the processor) and are defined in such a manner as to ensure that these bits will be zero when read while allowing several different hardware implementations. The writing of any value other than zero into these bits by the software shall cause undefined operation with respect to the subsequent value of these bits. The processor shall always provide zero when reading these bits (assuming that the previous write into these bits consisted of zero). This definition allows the following hardware implementations:

   1. The unused bits are copied into register locations when the registers are written and then read out on register reads.

   2. The unused bits are not implemented in hardware and zeros are provided when the registers are read with regeneration of parity.

b. UNUSED/ZERO* - These bits are identical to the above definition except that some systems have the following implementation:

   1. These unused bits are not implemented in hardware and zeros are provided along with the original parity when the registers are read. This implementation can in effect cause parity errors on the read when the unused bits were not written as zeros.

c. UNDEFINED - These bits have no specified state.

d. "___" - The operation indicated by the column is not applicable to the entry for this row.

e. COPY - These bits copy out on a read whatever was last written into them.

f. ONES - These bits will always be ones on a read regardless of previous writes.

g. ZEROS - These bits will always be zeros on a read regardless of previous writes.

h. REQ-ZERO - These bits are required to be zero or the processor operation will be undefined.

	COPY TO STATE REGISTER	COPY FROM STATE REGISTER	WRITE VIA MAINT. ACCESS	READ VIA MAINT. ACCESS	STORE INTO EXCHANGE PACKAGE	READ FROM EXCHANGE PACKAGE	STORE INTO SFSA (CALL/TRAP)	READ FROM SFSA (RETURN)	PROCESS INTERPRETATION RE BIT(S)
PROCESSOR STATE REGISTERS									
KEYPOINT BUFFER POINTER	—	—	—	—	—	—	—	—	—
BITS 0-15	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	—	—	IGNORED
BITS 61-63	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	—	—	ZERO
JOB PROCESS STATE (JPS) BIT 32	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	—	—	IGNORED
BITS 60-63	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	—	—	ZERO
MONITOR PROCESS STATE (MPS) BIT 32	—	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	—	—	IGNORED
BITS 60-63	—	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	—	—	ZERO
PAGE TABLE ADDRESS BIT 32	—	UNUSED/ZERO*	UNUSED/ZERO*	UNUSED/ZERO*	—	—	—	—	IGNORED
BITS 33-51	—	COPY	COPY	COPY	—	—	—	—	SEE 2.5.1.3
BITS 52-63	—	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	—	—	SEE 2.5.1.3
VMCL BITS 50-63	—	ZERO	—	ZERO	—	—	—	—	ZERO
PMF REG 22 BIT 1	—	—	UNUSED/ZERO*	UNUSED/ZERO*	—	—	—	—	—
REG 22 BITS 3-23	—	—	UNUSED/ZERO*	UNUSED/ZERO*	—	—	—	—	—
BYTES 4,5 WHEN NOT USED	—	—	UNUSED/ZERO*	UNUSED/ZERO*	—	—	—	—	—
BYTES 6,7	—	—	UNUSED/ZERO*	UNUSED/ZERO*	—	—	—	—	—
LEFTMOST BIT OF BYTES 9,11,13,15	—	—	UNUSED/ZERO*	UNUSED/ZERO*	—	—	—	—	—
PROCESS STATE REGISTERS									
P REG BITS 0,1,8,9	—	ZERO	ZERO	ZERO	ZERO	ZERO	ZERO	ZERO	IGNORED
BIT 32,63	—	COPY	COPY	COPY	COPY	COPY	COPY	COPY	ASE, IF NON-ZERO
P REG BITS 2-7	REQ-ZERO	COPY	REQ-ZERO	COPY	COPY	REQ-ZERO	COPY	REQ-ZERO	SEE 2.1.1.1
USER MASK REGISTER BITS 0-6	ONES	ONES	ONES	ONES	ONES	ONES	ONES	ONES	ONES
UNTRANSLATABLE POINTER BITS 32,61-63	—	COPY	COPY	COPY	COPY	COPY	—	—	—
TRAP POINTER BITS 32,61-63	COPY	COPY	COPY	COPY	COPY	COPY	—	—	ASE, IF NON-ZERO
DEBUG LIST POINTER BITS 32,61-63	COPY	COPY	COPY	COPY	COPY	COPY	—	—	ASE, IF NON-ZERO
A REGISTERS BIT 32	—	—	—	—	COPY	COPY	COPY	COPY	ASE, IF NON-ZERO
SEGMENT TABLE ADDRESS BIT 32	—	UNUSED/ZERO*	UNUSED/ZERO*	UNUSED/ZERO*	UNUSED/ZERO*	UNUSED/ZERO*	—	—	IGNORED
BIT 61,62,63	—	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	UNUSED/ZERO	—	—	ZERO
EXCHANGE PACKAGES									
C180 EXCHANGE PACKAGE - C180 PROCESS UNUSED BITS LISTED IN PARAGRAPH 2.5.2a	—	—	—	—	UNDEFINED	IGNORED	—	—	—
C180 EXCHANGE PACKAGE - C170 PROCESS UNUSED BITS DESCRIBED IN SECTION 7.4	—	—	—	—	UNDEFINED	IGNORED	—	—	—
C170 EXCHANGE PACKAGE UNUSED BITS DESCRIBED IN SECTION 7.5	—	—	—	—	ZERO	IGNORED	—	—	—
STACK FRAME SAVE AREAS									
SFSA - C180 PROCESS - TRAP UNUSED BITS DESCRIBED IN PARAGRAPH 2.5.4.1	—	—	—	—	—	—	UNDEFINED	IGNORED	—
SFSA - C180 PROCESS - CALL UNUSED BITS DESCRIBED IN PARAGRAPH 2.5.4.1	—	—	—	—	—	—	UNDEFINED	IGNORED	—
SFSA - C170 PROCESS - TRAP UNUSED BITS DESCRIBED IN SECTION 7.4	—	—	—	—	—	—	UNDEFINED	IGNORED	—

Table 2.1-2. Unused Bits

2.1.3.6 Nomenclature

Throughout the instruction descriptions contained in this specification, the following conventions shall be used with respect to nomenclature.

a. The expressions "Register Aj" and "the Aj Register" shall be used interchangeably to denote the 48-bit A Register specified by the 4-bit j field from an instruction. Thus, "Aj" shall denote one of the sixteen A Registers, A0 through AF (in hexadecimal notation) corresponding to j field values of 0 through 15 (in decimal notation), respectively.

   The 4-bit k field from an instruction shall be interpreted in a manner identical to the j field (as just described) with respect to the interchangeable expressions "Register Ak" and "the Ak Register."

b. The expressions "Register Xj" and "the Xj Register" shall be used interchangeably to denote the 64-bit X Register specified by the 4-bit j field from an instruction. Thus, "Xj" shall denote one of the sixteen X Registers, X0 through XF (in hexadecimal notation) corresponding to j field values of 0 through 15 (in decimal notation), respectively.

   The 4-bit k field from an instruction shall be interpreted in a manner identical to the j field (as just described) with respect to the interchangeable expressions "Register Xk" and "the Xk Register."

c. With respect to the X Registers, the terms "Left" and "Right" shall be used to denote the leftmost and rightmost 32-bit positions, respectively. Thus, "Register Xk Left" shall denote the leftmost 32-bit positions, 00 through 31, of the Xk Register and "Register Xk Right" shall denote the rightmost 32-bit positions, 32 through 63, of the Xk Register.

d. Parentheses shall be used within instruction names to denote "the contents of."

e. Units of information shall be referred to as bytes (8 bits), parcels (16 bits), halfwords (32 bits) or words (64 bits) with the following numbering conventions (always numbered consecutively from left to right):

Bits	00 ⟶	08 ⟶	16 ⟶	24 ⟶	32 ⟶	40 ⟶	48 ⟶	56 ⟶
Bytes	0	1	2	3	4	5	6	7
Parcels	0		1		2		3	
Halfwords	0				1			
Word	0							
	00 ---⟶ 63							

Note: Alphanumeric (including decimal) and floating point data formats are illustrated in sections 2.3 and 2.4, respectively of this specification.

f. Bits within registers are numbered consecutively from left to right with the rightmost bit always equal to 63. *For CYBER 170 bit numbering, see 7.2.1.*

2.1.4 Address Arithmetic

Address arithmetic operations, referred to as "indexing" and "displacement," shall be performed on signed, 32-bit integers using two's complement addition without overflow detection.

2.1.5 Address Exception

When the leftmost bit of the BN field, (position 32), in any PVA is equal to a one at the time it is used to access central memory, an Address Specification error shall be recorded, the central memory access shall be inhibited, and the corresponding program interruption shall occur. See subparagraph 2.8.1.5 of this specification.

2.1.6 Instruction Reference Numbers

Prior to the assignment of operation codes, each instruction was identified by a three-digit reference number. These reference numbers are shown in this specification now only for historical continuity.

Appendix A lists CP instructions in reference number sequence. All other tabulations, however, emphasize the operation code of the instruction, which has become the preferred instruction identifier.

2.1.7 Zero Field Length

The following instructions make memory references controlled by a field length. When the controlling field length is zero, these instructions are no-ops. All of the normal address exception detection for data fields (including the field with zero field length) may but need not be performed. This address exception detection includes: Access Violation, Invalid Segment, Address Specification Error, and Page Fault as described in section 2.8.1.

The normal debug scan for addresses associated with data fields (including the field with zero field length) shall be performed as per section 2.7.2.

Load/Store Multiple (Op. 80, 81)
Decimal Sum, Difference, Product, Quotient (Op. 70, 71, 72, 73)
Decimal Scale and Scale Rounded (Op. E4, D5)
Decimal Compare (Op. 74)
Numeric Move (Op. 75)
Byte Compare and Compare Collated (Op. 77, E9)
Byte Scan While Nonmember (Op. F3)
Byte Translate (Op. EB)
Move Bytes (Op. 76)
Move, Compare, Add Immediate Data (Op. F9, FA, FB)
All Vector Instructions (Op. 4X-5X)

2.2 GENERAL INSTRUCTIONS

For the purpose of this specification, the instructions comprising the General Instruction group shall be further classified, according to function, as described by the titles for paragraph numbers 2.2.1 through 2.2.10 of this specification.

2.2.1 Load and Store

This subgroup of instructions shall provide the means for transferring data, in the form of a single bit, a byte string, a 64-bit word, or multiple 64-bit words between one or more registers and one or more locations in central memory as specified by the individual operation codes.

For the purpose of establishing operand access validity for the associated central memory read and write accesses, the ring number used for validation shall be the value of the ring number contained in bit positions 16 through 19 of the associated A Register.

The central memory operand access type for the Load Bytes to Xk from (P) displaced by Q, (Op. 86) shall be execute-access (see subparagraph 2.2.1.4). For all other load and store instructions in this subgroup, the central memory operand access types shall be read-access for any instruction which loads an A or X register, and write-access for any instruction which stores an A or X register.

Instructions which transfer data from one or more registers to central memory, (namely, Store instructions), shall not alter the contents of any register which serves as a source of the data to be transferred to central memory.

2.2.1.1 Load/Store Bytes, Xk; Length Per S

a. Load Bytes to Xk from (Aj displaced by D and indexed by XiR), Length per S

DSjkiD (Ref. 001)

b. Store Bytes from Xk at (Aj displaced by D and indexed by XiR), Length per S

DSjkiD (Ref. 003)

Operation: These instructions shall transfer a field of bytes between Register Xk and a byte field in central memory with the direction of transfer determined by the operation code. The length of the byte field in central memory shall be determined from the instruction's S field as follows:

Load Bytes...	S = 0-7	Length = 1-8
Store Bytes...	S = 8-F	Length = 1-8

The bytes in Register Xk shall be right-justified, so that the appropriate leftmost byte positions in Register Xk shall be cleared for load instructions with lengths less than eight, and the appropriate leftmost byte positions within the Xk Register shall not be transferred for store instructions with lengths less than eight.

Addressing: The beginning (the leftmost byte position) of the byte string in central memory shall be determined by means of the PVA obtained from the Aj Register, modified by byte item counts as follows:

Displacement and Indexing: The 32-bit halfword obtained from register Xi Right and the 32-bit quantity obtained by left-extending the D field with zeros shall be added to the rightmost 32 bits of the PVA obtained from the Aj Register. In this context, the contents of the X0 Register shall be interpreted as consisting of all zeros.

2.2.1.2 Load/Store Word, Xk

a. Load Xk from (Aj displaced by 8*D and indexed by 8*XiR)

A2jkiD (Ref. 005)

b. Load Xk from (Aj displaced by 8*Q)

82jkQ (Ref. 006)

c. Store Xk at (Aj displaced by 8*D and indexed by 8*XiR)

A3jkiD (Ref. 007)

d. Store Xk at (Aj displaced by 8*Q)

83jkQ (Ref. 008)

Operation: These instructions shall transfer a word between Register Xk and a word location in central memory. The direction of transfer shall be determined by the operation code.

Addressing: The item location in central memory shall be determined by means of the PVA obtained from register Aj modified by a 32-bit quantity calculated as follows:

Displacement and Indexing: The 32-bit halfword obtained from register Xi Right shall be shifted left 3 bit positions, end-off with zeros inserted; the 12-bit quantity obtained from the D field of the instruction shall be expanded to 29 bits by extending zeros on the left and shall then be shifted left 3-bit positions with zeros inserted on the right. The two 32-bit quantities resulting from these operations shall then be added to the rightmost 32 bits of the PVA obtained from the Aj register. In this context, the contents of register X0 shall be interpreted as consisting of all zeros.

Displacement: The Q field from the instruction shall be expanded to 29 bits by means of sign extension and shall then be shifted left 3-bit positions with zeros inserted on the right. The 32-bit result shall then be added to the rightmost 32 bits of the PVA obtained from the Aj register.

Notes: Unless the PVA from the Aj register consists of a byte address which is 0 modulo 8, an Address Specification error shall be detected, the execution of the instruction shall be inhibited, and the corresponding program interruption shall occur. See subparagraph 2.8.1.5 of this specification.

2.2.1.3 Load/Store Bytes, Xk; Length Per X0

a. Load Bytes to Xk from (Aj displaced by D and indexed by XiR), Length Per X0

A4jkiD (Ref. 009)

b. Stores Bytes from Xk at (Aj displaced by D and indexed by XiR), Length Per X0

A5jkiD (Ref. 011)

Operation: These instructions shall transfer a field of bytes between Register Xk and a byte field in central memory with the direction of the transfer determined by the operation code. The length of the byte field in central memory shall be determined by adding one to the value of the rightmost 3 bits contained in Register X0.

In all other respects, these operations shall be identical to those described in subparagraph 2.2.1.1 of this specification.

2.2.1.4 Load Bytes Xk; Length Per j

Load Bytes to Xk from (P displaced by Q), Length per j

86jkQ (Ref. 013)

Operation: This instruction shall transfer a field of bytes from central memory to register Xk. The length of the byte field in central memory shall be determined by adding one to the value of the rightmost 3 bits of the j field from the instruction. The byte(s) loaded into Register Xk shall be right-justified so that the appropriate leftmost byte position(s) in Register Xk shall be cleared for lengths less than eight.

Addressing: The beginning (the leftmost byte position) of the byte field in central memory shall be determined by expanding the Q field to 32 bits by means of sign extension and then adding the result to the rightmost 32 bits of the PVA obtained from the P Register.

Notes: The read operation for the field of bytes from central memory shall be tested for access validity as if it were an instruction fetch, thus requiring execute-access rather than read-access validity.

The read operation for the field of bytes from central memory shall not be tested for read access. Testing for execute access could be tested but need not since instruction execution already has tested this.

2.2.1.5 Load/Store Bit, Xk

a. Load Bit to Xk from (Aj displaced by Q and bit-indexed by X0R)

88jkQ (Ref. 014)

b. Store Bit from Xk at (Aj displaced by Q and bit-indexed by X0R)

89jkQ (Ref. 015)

Operation: These instructions shall transfer a single bit between Register Xk Right, bit position 63, and a bit position in central memory, with the direction of the transfer determined by the operation code. Additionally, the load instruction shall clear the Xk Register in its leftmost 63 bit positions, 00 through 62.

Addressing: The byte in central memory, containing the bit position to be loaded from or stored into, shall be addressed by means of the PVA contained in the Aj Register modified as follows: The 32-bit halfword obtained from Register X0 Right shall be shifted right three bit positions, end-off with sign extension on the left, and the Q field from the instruction shall be expanded to 32 bits by means of sign extension. These two 32-bit results shall then be added to the rightmost 32 bits of the PVA obtained from the Aj Register.

Bit Selection: The bit position within the addressed byte in central memory shall be selected by means of the rightmost three bits obtained from Register X0 Right, bit positions 61 through 63. Values from 0 through 7 for these three bits shall select the corresponding bit position, 0 through 7 within the central memory byte.

Notes: The instruction which transfers a bit to central memory shall accomplish the associated central memory operations in a nonpreemptive manner, i.e., the byte containing the bit to be stored shall be read, modified in the appropriate bit position to the extent required, and then written such that no other accesses from any port to the addressed byte shall be permitted between these read and write accesses. When clearing a synchronization "lock" with this instruction, preserialization is required. This should be achieved by issuing a "Test and Set Bit" instruction (124) immediately prior to the Store Bit. Since the 124 instruction postserializes, this sequence effectively preserializes the clearing of the "lock."

For the instruction which transfers a bit to central memory, operand access validation shall consist of write access validation only.

2.2.1.6 Load/Store Address, Ak

a. Load Ak from (Aj displaced by D and indexed by XiR)

A0jkiD (Ref. 016)

b. Load Ak from (Aj displaced by Q)

84jkQ (Ref. 017)

c. Store Ak at (Aj displaced by D and indexed by XiR)

A1jkiD (Ref. 018)

d. Store Ak at (Aj displaced by Q)

85jkQ (Ref. 019)

Operation: These instructions shall transfer six bytes between the Ak register, right-justified, and a six byte field in central memory, with the direction of transfer and the addressing of central memory determined by the operation code.

Addressing: For the A0 and A1 instructions, the leftmost byte position of the six byte field in central memory shall be addressed by means of the PVA initially contained in register Aj, modified by byte item counts in a manner identical to that described in section 2.2.1.1.

For the 84 and 85 instructions, the leftmost byte position of the six byte field in central memory shall be addressed by means of the PVA initially contained in Register (Aj), modified in its rightmost 32-bit positions by the addition of the 32-bit quantity obtained by left extending the sign of the 16-bit Q field from the associated instruction.

Special Load Conditions: The instructions which load Register Ak shall unconditionally transfer only the rightmost 44 bits of the six byte field from central memory to bit positions 20 through 63 of Register Ak.

When the instructions which load Ak are executed, the larger value of 1) the leftmost 4 bits of the six byte field from central memory, 2) the leftmost 4 bits in bit positions 16 through 19 of the Aj Register, and 3) the R1 field contained in the 4-bit positions 08 through 11 of the segment descriptor associated with the PVA obtained from Register Aj, shall be transferred to bit positions 16 through 19 of Register Ak.

(For the format of a segment descriptor and the definition of its R1 field, see paragraphs 3.3.1 and 3.6.2 of this specification, respectively.)

Ring Number Zero Test: [See section 1.6 for a list of the C180 models that do not perform this test.] When the leftmost 4 bits of the six byte field from central memory are all equal to zero, a Ring Number Zero condition shall be detected and, following the completion of the associated Load instruction's execution, the corresponding program interruption shall take place. See subparagraph 2.8.1.13 of this specification.

2.2.1.7 Load/Store Multiple

a. Load Multiple Registers from (Aj displaced by 8*Q), Selectivity Per XkR

80jkQ (Ref. 020)

b. Store Multiple Registers at (Aj displaced by 8*Q), Selectivity Per XkR

81jkQ (Ref. 021)

Operation: These instructions shall transfer data between the general registers and central memory with the direction of the transfer determined by the operation code. Central memory address formation and general register selections shall be performed as follows:

Address Formation. The beginning address in central memory, of the contiguous word locations to which, or, from which, as determined by the operation code, the designated transfers shall take place, shall be formed by means of displacement addressing. The 16-bit Q field from the instruction shall be expanded to 29 bits by means of sign extension, these 29 bits shall be shifted left three bit positions with zeros inserted on the right, and this 32-bit shifted result shall be added to the rightmost 32 bits of the PVA initially contained in the Aj Register. The resulting PVA shall be used as the beginning address of the word field in central memory referenced by these instructions.

Register Selection. Selectivity of transfers between general registers and central memory shall be accomplished by interpreting the rightmost 16 bits initially contained in Register Xk Right as four fields of 4 bits each in the following manner:

```
|<------------- Register Xk Right ------------->|
|48        51|52        55|56        59|60        63|
|     4      |     4      |     4      |     4      |

      ↑           ↑           ↑           ↑
      ↑           ↑           ↑          Number of the last X Register
      ↑           ↑           ↑          to be transferred
      ↑           ↑           ↑
      ↑           ↑          Number of the last A Register
      ↑           ↑          to be transferred
      ↑           ↑
      ↑          Number of the first X Register to be transferred
      ↑
     Number of the first A Register to be transferred
```

When the first register number is greater than the associated last register number, none of the registers from the corresponding A or X Register groups shall be loaded or stored.

Transfers between registers and central memory shall begin with the A Register Group and end with the X Register Group to the extent that the Registers within these groups are designated by the rightmost 16 bits of Register Xk Right. A positive offset, applied to the PVA designating the first word location of the central memory field, shall begin with zero and shall be increased by eight for each designated transfer as it is accomplished during the course of instruction execution.

The relationship between the bits contained in positions 48 through 63 of Register Xk Right, the 16 registers contained in each of the general register groups A and X, and the positive offset values applied to the beginning address of the word field in central memory, are illustrated in figure 2.2-1 for the case in which all 32 Registers are transferred.

The leftmost 16 bits of the word locations in the central memory field, which are associated with A Registers to the extent designated, shall not be used by the instruction which loads multiple registers and shall be cleared by the instruction which stores multiple registers.

Special Load A Conditions: The instruction which loads A Registers shall unconditionally transfer only the rightmost 44-bit positions 20 through 63 of each appropriate word from central memory to the corresponding bit positions of the designated A Registers.

As part of the execution of the Load Multiple instruction, the larger value of 1) the 4 bits in bit positions 16 through 19 of each appropriate word from central memory, 2) the leftmost 4 bits in bit positions 16 through 19 of the Aj Register, and 3) the R1 field contained in the 4-bit positions 08 through 11 of the segment descriptor associated with the PVA obtained from Register Aj, shall be transferred to bit positions 16 through 19 of each of the appropriately designated A Registers.

Ring Number Zero Test: [See section 1.6 for a list of the C180 models that do not perform this test.] With respect to the designated A Registers, when all 4 bits in positions 16 through 19 of any associated word from central memory are equal to zero, a Ring Number Zero condition shall be detected and, following the completion of the Load Multiple instruction's execution, the associated program interruption shall occur. See subparagraph 2.8.1.13 of this specification.

Notes: For both of these operation codes, unless the PVA initially contained in the Aj Register consists of a byte address which is equal to 0, modulo 8, an Address Specification error shall be detected, all transfers associated with the execution of these instructions shall be inhibited, and the corresponding program interruption shall occur. See subparagraph 2.8.1.5 of this specification.

"For the instruction which loads multiple registers, (reference number 20), the PVA resulting from the addition of (Aj) and the Q field from the instruction, shall constitute the only Data Read argument for this instruction with respect to a Debug List Scan operation. (See paragraph 2.7.2.)

For the instruction which stores multiple registers, (reference number 21), the PVA resulting from the addition of (Aj) and the Q field from the instruction shall constitute the only Data Write argument for the instruction with respect to a Debug List Scan operation. (See paragraph 2.7.2.)

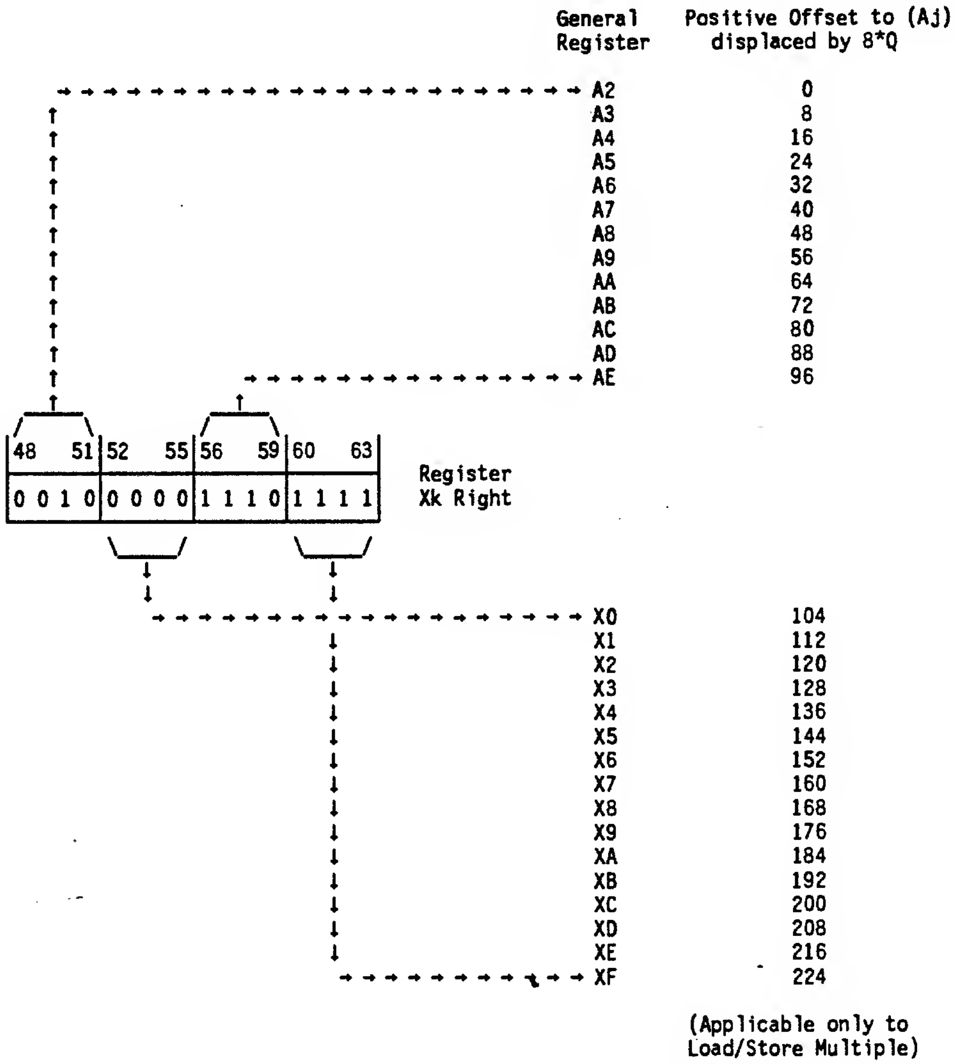


Figure 2.2-1. Register Selectivity Correspondence

2.2.2 Integer Arithmetic

Integer arithmetic operations shall be performed on words and halfwords contained in Register Xk and Register Xk Right, respectively, as described in the following subparagraphs.

Binary integers contained in the X Registers shall consist of signed, two's complement, 32-bit or 64-bit quantities. The leftmost bit, (in position 00 for 64-bit integers and in position 32 for 32-bit integers), shall constitute the sign bit. Positive quantities shall consist of a sign bit in the zero state with the 31 or 63 contiguous bits immediately to the right of the sign bit, expressing the magnitude of the number. Negative quantities shall be expressed as the two's complement of their positive representations, resulting in a sign bit in the one state. Conceptually, the two's complement of a binary integer shall be formed by adding one to its one's complement representation. (Conceptually, the one's complement of a binary integer shall be formed by subtracting it, bit-for-bit, from another number consisting entirely of one bits).

00	01 63
S	

Register Xk: 64-bit integer

00 31	32	33 63
///////////////	S	

Register Xk Right: 32-bit integer

The ranges in magnitude, M, covered by binary integers in each of the two fixed point formats, shall be as follows:

32-bit Integer: $-2^{31} \leq M \leq 2^{31}-1$ 64-bit Integer: $-2^{63} \leq M \leq 2^{63}-1$

2.2.2.1 Integer Sum, Xk

a. Integer Sum, Xk replaced by Xk plus Xj

24jk (Ref. 022)

b. Integer Sum, Xk replaced by Xj plus Q

8BjkQ (Ref. 143)

c. Integer Sum, Xk replaced by Xk plus j

10jk (Ref. 166)

These instructions shall obtain a 64-bit addend from the initial contents of Register Xj, or from the 16-bit sign-extended Q field of the instruction, or from the 4-bit zeros extended j field of the instruction, as determined by the operation code. The 64-bit addend thus derived shall be added to the 64-bit word initially contained in Register Xk or Xj, as correspondingly determined by the operation code, and shall transfer the 64-bit sum to Register Xk. Each 64-bit word shall be treated as a signed two's complement integer.

When the augend and addend are identically signed, and their addition produces a sum with a sign opposite that of the addend and augend, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

2.2.2.2 Integer Difference, Xk

a. Integer Difference, Xk replaced by Xk minus Xj

25jk (Ref. 023)

b. Integer Difference, Xk replaced by Xk minus j

11jk (Ref. 167)

These instructions shall obtain a 64-bit subtrahend from the initial contents of Register Xj or from the 4-bit zeros extended j field of the instruction, as determined by the operation code. The 64-bit subtrahend thus derived shall be subtracted from the 64-bit word initially contained in Register Xk and the difference shall be transferred to Register Xk. Each 64-bit word shall be treated as a signed two's complement integer.

When the minuend and subtrahend are oppositely signed and the subtraction produces a difference with a sign opposite that of the minuend, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

2.2.2.3 Integer Product, Xk

a. Integer Product, Xk replaced by Xk times Xj

26jk (Ref. 024)

b. Integer Product, Xk replaced by Xj times Q

B2jkQ (Ref. 168)

These instructions shall obtain a 64-bit multiplier from the initial contents of Register Xj, or from the 16-bit sign-extended Q field of the instruction, as determined by the operation code. The 64-bit multiplier thus derived shall be taken times the 64-bit word initially contained in Register Xk or Register Xj as determined by the operation code. The result of this multiplication shall consist of a 128-bit intermediate product, algebraically signed. The rightmost 64 bits of this intermediate product shall be transferred to the Xk Register.

Unless the leftmost 65 bits of the properly signed intermediate product are all in the same state, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

2.2.2.4 Integer Quotient, Xk

Integer Quotient, Xk replaced by Xk divided by Xj

27jk (Ref. 025)

This instruction shall divide the 64-bit word initially contained in the Xk Register by the 64-bit word initially contained in the Xj Register. Provided the divisor is not equal to zero, the results of the division, consisting of a 64-bit quotient algebraically signed, shall be transferred to Register Xk.

When the divisor is equal to zero, the contents of Register Xk shall not change and a Divide Fault condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.8 of this specification.

For the case in which -2^{63} is divided by -2^{0}, the quotient result shall have the form of -2^{63}, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

Note: The division shall produce a quotient result which, in its absolute form, shall not have been rounded upwards. Thus, when the absolute value of the quotient result is concatenated to a single zero bit, that quantity shall be equal to or less than the absolute value of the quotient computed to one additional bit of precision in the rightmost position. Moreover, when the absolute value of the quotient result is increased by one and concatenated to a single zero bit, that quantity shall be greater than the absolute value of the quotient computed to one additional bit of precision in the rightmost position.

2.2.2.5 Half Word Integer Sum, XkR

a. Integer Sum, XkR replaced by XkR plus XjR

20jk (Ref. 027)

b. Integer Sum, XkR replaced by XjR plus Q

8AjkQ (Ref. 028)

c. Integer Sum, XkR replaced by XkR plus j

28jk (Ref. 029)

Operation: These instructions shall obtain a 32-bit addend from the initial contents of Register Xj Right, from the 16-bit sign extended Q field of the instruction, or from the 4-bit zeros extended j field of the instruction, as determined by the operation code.

The 32-bit addend thus derived, shall be added to the 32-bit halfword initially contained in Register Xk Right or Register Xj Right, as determined by the operation code and the sum shall be transferred to Register Xk Right. Each of these 32-bit halfwords shall be treated as signed two's complement integers.

When the augend and addend are identically signed, and their addition produces a sum with a sign opposite that of the addend and augend, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

2.2.2.6 Half Word Integer Difference, XkR

a. Integer Difference, XkR replaced by XkR minus XjR

21jk (Ref. 030)

b. Integer Difference, XkR replaced by XkR minus j

29jk (Ref. 031)

Operation: These instructions shall obtain a 32-bit subtrahend from the initial contents of Register Xj Right or from the 4-bit zeros extended j field from the instruction, as determined by the operation code.

The 32-bit subtrahend thus derived shall be subtracted from the 32-bit halfword initially contained in Register Xk Right and the difference shall be transferred to Register Xk Right. Each of these 32-bit halfwords shall be treated as signed two's complement integers. When the minuend and subtrahend are oppositely signed and the subtraction produces a difference with a sign opposite that of the minuend, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

2.2.2.7 Half Word Integer Product, XkR

a. Integer Product, XkR replaced by XkR times XjR

22jk (Ref. 032)

b. Integer Product, XkR replaced by XjR times Q

8CjkQ (Ref. 033)

These instructions shall obtain a 32-bit multiplier from the initial contents of Register Xj Right or from the 16-bit sign extended Q field of the instruction, as determined by the operation code.

The 32-bit multiplier thus derived shall be taken times the 32-bit halfword initially contained in Register Xk Right or Register Xj Right as determined by the operation code. The result of the multiplication shall consist of a 64-bit intermediate product, algebraically signed. The rightmost 32 bits of this intermediate product shall be transferred to Register Xk Right.

Unless the leftmost 33 bits of the properly signed intermediate product are all in the same state, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

2.2.2.8 Half Word Integer Quotient, XkR

Integer Quotient, XkR replaced by XkR divided by XjR

23jk (Ref. 034)

This instruction shall divide the 32-bit halfword initially contained in Register Xk Right by the 32-bit halfword initially contained in Register Xj Right. Provided the divisor is not equal to zero, the results of the division, consisting of a 32-bit quotient, algebraically signed, shall be transferred to Register Xk Right.

When the divisor is equal to zero, the contents of Register Xk shall not be changed and a Divide Fault condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.8 of this specification.

For the case in which -2^{31} is divided by -2^{0}, the quotient result shall have the form of -2^{31}, an Arithmetic Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

Note: The division shall produce a quotient result which, in its absolute form, shall not have been rounded upwards. Thus, when the absolute value of the quotient result is concatenated to a single zero bit, that quantity shall be equal to or less than the absolute value of the quotient computed to one additional bit of precision in the rightmost position. Moreover, when the absolute value of the quotient result is increased by one and concatenated to a single zero bit, that quantity shall be greater than the absolute value of the quotient computed to one additional bit of precision in the rightmost position.

2.2.2.9 Integer Compare

a. Integer Compare, Xj to Xk, result to X1R

2Djk (Ref. 035)

b. Integer Compare, XjR to XkR, result to X1R

2Cjk (Ref. 036)

Operation: These instructions shall perform an algebraic comparison of the signed, two's complement, binary integer initially contained in Register Xj to the signed, two's complement, binary integer initially contained in Register Xk. These compared values shall consist of 64 bits or 32 bits (right-justified in positions 32 through 63) as determined by the operation code. In this context the contents of the X0 Register shall be interpreted as consisting entirely of zeros.

Results: When the comparison finds these quantities equal, Register X1 Right shall be cleared in all 32 bit positions. When the comparison finds the quantity obtained from Register Xj greater than the quantity obtained from Register Xk, Register X1 Right shall be cleared in bit positions 32 and 34 through 63, and shall be set in bit position 33. When the comparison finds the quantity obtained from Register Xj less than the quantity obtained from Register Xk, Register X1 Right shall be cleared in bit positions 34 through 63 and shall be set in bit positions 32 and 33.

2.2.3 Branch

The instructions within this subgroup shall consist of both conditional and unconditional branch instructions.

Each conditional branch instruction shall perform a comparison between the contents of two general registers. Then, based on the relationship between the results of that comparison and the branch condition as specified by means of the instruction's operation code, each conditional branch instruction shall perform either a normal exit or a branch exit.

Normal exit: When the results of a comparison do not satisfy the branch condition as specified by the operation code, a normal exit shall be performed. A normal exit for all conditional branch instructions shall consist of adding four to the rightmost 32 bits of the PVA obtained from the P Register with that 32-bit sum returned to the P Register in its rightmost 32-bit positions.

Branch exit: When the results of a comparison satisfy the branch condition as specified by the operation code, a branch exit shall be performed. A branch exit shall consist of expanding the 16-bit Q field from the instruction to 31 bits by means of sign extension, shifting these 31 bits left one bit position with a zero inserted on the right, and adding this 32-bit shifted result to the rightmost 32 bits of the PVA obtained from the P Register with the 32-bit sum returned to the P Register in its rightmost 32-bit positions.

Unconditional branch instructions shall perform branch exits according to the appropriate instruction descriptions contained in subparagraphs 2.2.3.5 and 2.2.3.6 of this specification.

2.2.3.1 Conditional, X

a. Branch to P displaced by 2*Q if Xj equal to Xk

94jkQ (Ref. 037)

b. Branch to P displaced by 2*Q if Xj not equal to Xk

95jkQ (Ref. 038)

c. Branch to P displaced by 2*Q if Xj greater than Xk

96jkQ (Ref. 039)

d. Branch to P displaced by 2*Q if Xj greater than or equal to Xk

97jkQ (Ref. 040)

Each of these instructions shall perform an algebraic comparison of the 64-bit word obtained from Register Xj to the 64-bit word obtained from Register Xk. Each of these 64-bit words shall be treated as signed, two's complement, binary integers. The contents of Register X0 shall be interpreted as consisting entirely of zeros.

These instructions shall perform a normal exit or a branch exit in the manner previously described in paragraph 2.2.3 of this specification.

2.2.3.2 Conditional, X Right

a. Branch to P displaced by 2*Q if XjR equal to XkR

90jkQ (Ref. 041)

b. Branch to P displaced by 2*Q if XjR not equal to XkR

91jkQ (Ref. 042)

c. Branch to P displaced by 2*Q if XjR greater than XkR

92jkQ (Ref. 043)

d. Branch to P displaced by 2*Q if XjR greater than or equal to XkR

93jkQ (Ref. 044)

Each of these instructions shall perform an algebraic comparison of the 32-bit halfword obtained from Register Xj Right with the 32-bit halfword obtained from Register Xk Right. Each of these 32-bit halfwords shall be treated as signed, two's complement, binary integers. The contents of Register X0 shall be interpreted as consisting entirely of zeros.

These instructions shall perform a normal exit or a branch exit in the manner previously described in paragraph 2.2.3 of this specification.

2.2.3.3 Branch and Increment

Branch to P displaced by 2*Q and increment Xk if Xj greater than Xk

9CjkQ (Ref. 045)

This instruction shall perform an algebraic comparison of the 64-bit word initially contained in Register Xj with the 64-bit word initially contained in Register Xk. Each of these 64-bit words shall be treated as signed, two's complement, binary integers. With respect to the Xj Register only, Register X0 shall be interpreted as consisting entirely of zeros.

When this comparison does not find the value initially contained in Register Xj greater than the value initially contained in Register Xk, a normal exit shall be performed in the manner previously described in paragraph 2.2.3 of this specification.

When this comparison finds the value initially contained in Register Xj greater than the value initially contained in Register Xk, a branch exit shall be performed in the manner previously described in paragraph 2.2.3 of this specification. In addition, the 64-bit word initially contained in Register Xk shall be increased by one in value and the result returned to the Xk Register. Overflow will be ignored.

2.2.3.4 Branch on Segments Unequal

Branch to P displaced by 2*Q if segments unequal; else compare byte numbers, result to X1R

9DjkQ (Ref. 046)

This instruction shall perform a bit-for-bit comparison between the 12-bit SEG field contained in bit positions 20 through 31 of Register Aj and the 12-bit SEG field contained in bit positions 20 through 31 of Register Ak. When the comparison finds the SEG fields not equal, this instruction shall perform a branch exit in the manner described in paragraph 2.2.3 of this specification.

When the comparison finds the SEG fields equal, this instruction shall perform an algebraic comparison of the 32-bit BN field contained in bit positions 32 through 63 of Register Aj to the 32-bit BN field contained in bit positions 32 through 63 of Register Ak and shall perform a normal exit in the manner described in paragraph 2.2.3 of this specification.

The algebraic comparison of the BN fields shall treat each of these 32-bit quantities as signed two's complement binary integers and shall store the result of their comparison into Register X1 Right as follows: when the BN fields are equal, Register X1 Right shall be cleared in all 32-bit positions.

When the BN field from Register Aj is greater than the BN field from Register Ak, Register X1 Right shall be cleared in bit positions 32 and 34 through 63, and shall be set in bit position 33. When the BN field from Register Aj is less than the BN field from Register Ak, Register X1 Right shall be cleared in bit positions 34 through 63 and shall be set in positions 32 and 33.

2.2.3.5 Branch Relative

Branch to P indexed by 2*XkR

2Ejk (Ref. 047)

This instruction shall perform an unconditional branch exit by modifying the contents of the P Register in its rightmost 32-bit positions as follows:

The 32-bit halfword obtained from Register Xk Right shall be shifted left one bit position, end-off with a zero inserted on the right, and the 32-bit shifted result shall be added to the rightmost 32 bits initially contained in the P Register. This 32-bit sum shall be returned to the P Register in its rightmost 32-bit positions.

2.2.3.6 Intersegment Branch

Branch to Aj indexed by 2*XkR

2Fjk (Ref. 048)

In the absence of all associated Virtual Addressing Mechanism exceptions (other than a Page Table Search Without Find condition at the branch address) this instruction shall perform a branch exit by modifying the Key, SEG and BN fields contained in the P Register as follows:

The 12-bit Segment field, SEG, contained in bit positions 20 through 31 of Register Aj shall be transferred to the corresponding 12-bit positions of the P Register.

The 32-bit halfword obtained from Register Xk Right shall be shifted left one bit position, end-off with a zero inserted on the right, and the 32-bit shifted result shall be added to the rightmost 32 bits obtained from Register Aj in bit positions 32 through 63. (In this context, the contents of Register X0 shall be interpreted as consisting entirely of zeros.) This 32-bit sum shall be transferred to the rightmost 32-bit positions, 32 through 63, of the P Register.

The Key field initially contained in the P Register shall be altered according to the description contained in subparagraph 3.6.3.2 of this specification.

Notes: The P(RN) field shall not be changed by the execution of this instruction.

2.2.4 Copy

The instructions within this subgroup shall provide the means for accomplishing inter-register transfers to the extent defined by the following instruction descriptions.

2.2.4.1 Copy, Xk Replaced by Xj

0Djk (Ref. 049)

This instruction shall transfer the 64-bit word initially contained in Register Xj to the 64-bit positions of Register Xk.

2.2.4.2 Copy, Xk Replaced by Aj

0Bjk (Ref. 050)

This instruction shall transfer the 48 bits contained in Register Aj to the rightmost 48-bit positions, 16 through 63, of Register Xk. The leftmost 16-bit positions, 00 through 15, of Register Xk shall be cleared.

2.2.4.3 Copy, Ak Replaced by Aj

09jk (Ref. 051)

This instruction shall transfer the 48 bits contained in Register Aj to the 48-bit positions of Register Ak.

2.2.4.4 Copy, Ak Replaced by Xj

0Ajk (Ref. 052)

This instruction shall unconditionally transfer the rightmost 44 bits contained in positions 20 through 63, of Register Xj to the corresponding 44-bit positions of Register Ak. The 4-bit field having the larger value in bit positions 16 through 19 of the Xj Register or the P Register, shall be transferred to the corresponding 4-bit positions of the Ak Register.

2.2.4.5 Copy, XkR Replaced by XjR

0Cjk (Ref. 053)

This instruction shall transfer the 32-bit halfword initially contained in Register Xj Right to the 32-bit positions, 32 through 63, of Register Xk Right. The initial contents of Register Xk Left shall not be changed.

2.2.5 Address Arithmetic

The instructions within this subgroup shall provide the means for accomplishing address arithmetic to the extent defined by the following instruction descriptions.

2.2.5.1 Address Increment, Signed Immediate

Address Ak replaced by Aj plus Q

8EjkQ (Ref. 054)

This instruction shall transfer the leftmost 16 bits initially contained in bit positions 16 through 31 of Register Aj to the corresponding 16-bit positions of Register Ak. In addition, the 16-bit Q field from the instruction, expanded to 32 bits by means of sign extension, shall be added to the rightmost 32 bits initially contained in bit positions 32 through 63 of Register Aj and the 32-bit sum shall be transferred to the corresponding rightmost 32-bit positions of Register Ak.

2.2.5.2 Address Relative

Address Ak replaced by P plus 2*XjR plus 2*Q

8FjkQ (Ref. 055)

This instruction shall transfer the leftmost 16 bits contained in bit positions 16 through 31 of the P Register to the corresponding 16-bit positions of the Ak Register. In addition, the 16-bit Q field from the instruction shall be expanded to 31 bits by means of sign extension, these 31 bits shall be shifted left one bit position with a zero inserted on the right, and this 32-bit shifted result shall be added to the rightmost 32 bits obtained from the P Register. This 32-bit sum shall be added to the rightmost 32 bits obtained from Register Xj Right, shifted left one bit position with a zero inserted on the right, and the final result shall be transferred to the rightmost 32-bit positions, 32 through 63, of Register Ak. In this context, the contents of Register X0 shall be interpreted as consisting entirely of zeros.

2.2.5.3 Address Increment, Indexed

Address Ak replaced by Ak plus XjR

2Ajk (Ref. 056)

This instruction shall add the 32 bits contained in Register Xj Right to the rightmost 32 bits initially contained in bit positions 32 through 63 of Register Ak and shall return the 32-bit sum to the rightmost 32-bit positions of Register Ak.

2.2.5.4 Address Increment, Modulo

Address Ak replaced by Ai plus D per j

A7jkiD (Ref. 161)

This instruction shall transfer the leftmost 16 bits initially contained in bit positions 16 through 31 of Register Ai to the corresponding 16-bit positions of Register Ak. In addition, the 12-bit D field from the instruction, expanded to 32 bits by extending zeros on the left, shall be added to the rightmost 32 bits intially contained in bit positions 32 through 63 of Register Ai. The leftmost 29 bits of this 32-bit sum shall be transferred to bit positions 32 through 60 of Register Ak. A logical product (AND) between the rightmost 3 bits of this 32-bit sum and the rightmost 3 bits of the j field from the instruction shall be performed, with the 3-bit result of the logical operation transferred to bit positions 61 through 63 of Register Ak.

Note: The truth table for the bit-by-bit logical product (AND) operation is provided in subparagraph 2.2.8.1 of this specification.

2.2.6 Enter

The instruction within this subgroup shall provide the means for entering immediate operands, (consisting of logical quantities of signed, two's complement binary integers), into the X Registers to the extent defined by the following instruction descriptions.

2.2.6.1 Enter Immediate

a. Enter Xk with plus j

3Djk (Ref. 057)

b. Enter Xk with minus j

3Ejk (Ref. 058)

Operation: These instructions shall expand the 4-bit j field from the instruction to 64 bits by extending 60 zeros on the left and shall transfer this 64-bit result or the two's complement of this 64-bit result, as determined by the operation code, to the Xk Register.

2.2.6.2 Enter Xk, Signed Immediate

Enter Xk with sign extended Q

8DjkQ (Ref. 059)

This instruction shall expand the 16-bit Q field from the instruction to 64 bits by means of sign extension and shall transfer this 64-bit result to the Xk Register.

2.2.6.3 Enter X0 or X1, Immediate Logical

a. Enter X0 with logical jk

3Fjk (Ref. 060)

b. Enter X1 with logical jk

39jk (Ref. 164)

These instructions shall form a 64-bit result consisting of 4-bit k field from the instruction in bit positions 60 through 63, the 4-bit j field from the instruction in bit positions 56 through 59 and zeros in bit positions 00 through 55, and shall transfer this result to Register X0 or X1, as determined by the instruction code.

2.2.6.4 Enter Signs

Enter XkL with signs per j

1Fjk (Ref. 061)

The value of the rightmost 2 bits of the j field from the instruction shall be translated as follows:

a. If = 00, the 32-bit positions, 00 through 31, of Register Xk Left shall be cleared.
b. If = 01, the 32-bit positions, 00 through 31, of Register Xk Left shall be set.
c. If = 10 or 11, the sign bit in position 32 of Register Xk Right shall be transferred to all 32 bit positions, 00 through 31, of Register Xk Left.

2.2.6.5 Enter X0 or X1, Signed Immediate

a. Enter X0 with sign extended jkQ

B3jkQ (Ref. 169)

b. Enter X1 with sign extended jkQ

87jkQ (Ref. 165)

These instructions shall expand the 24-bit concatenation of the j, k and Q fields from the instruction, right-justified, to 64 bits by extension of the most significant bit of the j field through bits 00 - 39 inclusive, and shall transfer this 64-bit quantity to bits 00 - 63 of Register X0 or X1.

2.2.7 Shift

The instructions within this subgroup shall provide the means for shifting the initial contents of the Xj Register and transferring the result to the Xk Register, to the extent defined by the following descriptions.

All of the instructions within this subgroup shall derive the computed shift count in the following manner: the rightmost 8 bits of the D field from the instruction shall be added to the rightmost 8 bits initially contained in bit positions 56 through 63 of Register Xi Right and the 8-bit sum shall represent the computed shift count. Any overflow from the 8-bit sum is ignored. In this context, the contents of Register X0 Right shall be interpreted as consisting entirely of zeros.

The instructions within this subgroup shall interpret the computed shift count as follows: the sign bit in the leftmost position of the 8-bit computed shift count shall determine the direction of the shift. When the computed shift count is positive, (sign bit of zero), these instructions shall left shift. When the computed shift count is negative, (sign bit of one), these instructions shall right shift. For 32-bit quantities, shifts shall be from 0 - 31 bits left and from 1 - 32 bits right. For 64-bit quantities, shifts shall be from 0 - 63 bits left and from 1 - 64 bits right. Based on an 8-bit signed two's complement shift count, these shifts are as follows:

32-bit

0111 1111 : : : 0010 0000	→ Left Shift 0 - 31 (repeating)
0001 1111	Left Shift 31
:	:
0000 0000	Left Shift 0
1111 1111	Right Shift 1
:	:
1110 0000	Right Shift 32
1101 1111 : : : 1000 0000	→ Right Shift 1 - 32 (repeating)

64-bit

0111 1111 : : : 0100 0000	→ Left Shift 0 - 63
0011 1111	Left Shift 63
:	:
0000 0000	Left Shift 0
1111 1111	Right Shift 1
:	:
1100 0000	Right Shift 64
1011 1111 : : : 1000 0000	→ Right Shift 1 - 64

When these interpretations of the computed shift count result in an actual shift count of zero, the associated instructions shall transfer the initial contents of the Xj Register to the Xk Register and no shifting shall be performed.

2.2.7.1 Shift Circular

Shift Circular, Xk replaced by Xj, direction and count per XiR plus D

A8jkiD (Ref. 062)

This instruction shall shift the 64-bit word initially contained in Register Xj, with the direction and number of bit positions to be shifted determined by the computed shift count, and shall transfer the result to Register Xk. The computed shift count shall be derived and interpreted in the manner described in paragraph 2.2.7 of this specification.

This instruction shall shift circularly such that bits shifted out one end of the 64-bit word shall be transferred into bit positions which become unoccupied at the opposite end of the 64-bit word as a result of the shift.

2.2.7.2 Shift End-Off

a. Shift End-off, Xk replaced by Xj, direction and count per XiR plus D

A9jkiD (Ref. 063)

b. Shift End-off, XkR replaced by XjR, direction and count per XiR plus D

AAjkiD (Ref. 064)

Operation: These instructions shall shift the 64-bit word initially contained in Register Xj or the 32-bit halfword contained in Register Xj Right, as determined by the operation code, and shall transfer the result to Register Xk or Register Xk Right as correspondingly determined by the operation code. The direction and number of bit positions to be shifted shall be determined by the computed shift count. The computed shift count shall be derived and interpreted in the manner described in paragraph 2.2.7 of this specification.

Right Shift: Right shifts shall be performed end-off on the right and sign extended on the left. Thus, bits shifted out of the rightmost bit position shall be lost and the leftmost bit position, which would otherwise become unoccupied for each bit position shifted, shall be left unchanged.

Left Shift: Left shifts shall be performed end-off on the left with zeros inserted on the right. Thus, bits shifted out of the leftmost bit position shall be lost and the rightmost bit position, which becomes unoccupied for each bit position shifted, shall be cleared.

2.2.8 Logical

The instructions within this subgroup shall provide the means for performing Boolean operations on the 64-bit words contained in the X Registers to the extent defined by the following instruction descriptions.

2.2.8.1 Logical Sum, Difference, and Product

a. Logical Sum, Xk replaced by Xk or Xj

18jk (Ref. 065)

b. Logical Difference, Xk replaced by Xk EOR Xj

19jk (Ref. 066)

c. Logical Product, Xk replaced by Xk AND Xj

1Ajk (Ref. 067)

These instructions shall perform a logical operation between the 64-bit word initially contained in the Xj Register and the 64-bit word initially contained in the Xk Register and shall return the 64-bit Boolean result to the Xk Register.

The logical operations performed by these instructions shall consist of a logical sum (OR), a logical difference (EOR) or a logical product (AND), as determined by the operation code, and accomplished according to the following truth tables.

OR:		EOR:		AND:	
	0011		0011		0011
	<u>0101</u>		<u>0101</u>		<u>0101</u>
	0111		0110		0001

2.2.8.2 Logical Complement

Logical Complement, Xk replaced by Xj NOT

1Bjk (Ref. 068)

This instruction shall transfer the one's complement of the 64-bit word initially contained in the Xj Register to the 64-bit positions of the Xk Register.

Conceptually, taking the one's complement of a 64-bit word shall be accomplished by subtracting it, bit-for-bit, from a 64-bit word consisting entirely of one bits.

One's Complement Truth Table;	1's :	1111
	[Xj] :	<u>0110</u>
	[Xk] :	1001

2.2.8.3 Logical Inhibit

Logical Inhibit, Xk replaced by Xk AND Xj NOT

1Cjk (Ref. 069)

This instruction shall perform a logical product between the one's complement of the 64-bit word initially contained in the Xj register and the 64-bit word initially contained in the Xk register and shall return the 64-bit Boolean result to the Xk register.

The truth tables for the logical product and one's complement operations are provided in subparagraphs 2.2.8.1 and 2.2.8.2, respectively, of this specification.

2.2.9 Register Bit String

The instructions within this subgroup shall provide the means for addressing a contiguous string (field) of bits, beginning and ending independently with any bit positions within a 64-bit word.

For each of the instructions in this subgroup, the bit strings shall be addressed by means of a 12-bit field referred to as a bit string descriptor. This field of bits, including the field constituting a bit field descriptor, shall be numbered from left to right, with the leftmost bit numbered 00. The 6-bit subfield in bit positions 00 through 05 of a bit string descriptor shall designate the beginning, or leftmost, bit position within a 64-bit word. The 6-bit subfield in bit positions 06 through 11 of the bit string descriptor is a length designator that is interpreted as designating one less than the length (in bits) of a bit string within a 64-bit word.

00 05	06 11
Leftmost Position Designator	Length Designator (bit length -1)

Bit String Descriptor

For all instructions within this subgroup, indexing shall be carried out as follows: the bit string descriptor obtained from the D field of the instruction shall be zero-extended on the left to 32 bits and then added, without overflow detection, to the contents of register Xi Right (in this context, the contents of register X0 shall be interpreted as all zeros); the rightmost 12 bits of the result shall then be interpreted as a bit string descriptor, in the manner described above. For each of the instructions in this subgroup, when, after indexing, the sum of the "Leftmost Position Designator" and the "Length Designator" is greater than 63 (decimal), an Instruction Specification error shall be detected, the execution of the associated instruction shall be inhibited and the corresponding program interruption shall occur.

2.2.9.1 Isolate Bit Mask

Isolate Bit Mask into Xk per XiR plus D

ACjkiD (Ref. 070)

This instruction shall generate, in Xk, a bit mask consisting of a field of contiguous one bits whose leftmost and rightmost bit positions are determined by the bit field descriptor calculated and interpreted as specified in subparagraph 2.2.9. All bit positions to the left of the leftmost bit position and all bit positions to the right of the rightmost bit position (leftmost bit position plus length designator), if any, shall consist of zeros.

2.2.9.2 Isolate

Isolate into Xk from Xj per XiR plus D

ADjkiD (Ref. 071)

This instruction shall obtain a field of contiguous bits from the initial contents of the Xj register, shall clear all 64-bit positions of the Xk register, and shall then transfer that field of contiguous bits, right-justified, into the Xk register. The leftmost and rightmost bit positions of the field obtained from the Xj register shall be defined by the bit field descriptor calculated and interpreted as specified in subparagraph 2.2.9.

2.2.9.3 Insert

Insert into Xk from Xj per XiR plus D

AEjkiD (Ref. 072)

This instruction shall transfer a field of contiguous bits, initially contained right-justified in the Xj register, to a field of contiguous bit positions in the Xk register. The length of the bit string obtained from the Xj register, and the leftmost and rightmost bit positions of the Xk register shall be defined by the bit string descriptor calculated and interpreted as specified in paragraph 2.2.9. All bit positions to the left of the leftmost bit position, and all bit positions to the right of the rightmost bit position of the Xk register, if any, shall be left unchanged.

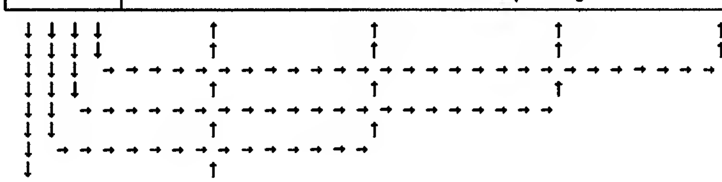
2.2.10 Mark to Boolean

Mark to Boolean, Set Xk per j and X1R

1Ejk (Ref. 145)

This instruction shall test the two bits initially contained in the leftmost two bit positions, 32 and 33, of Register X1 Right according to the 4-bit j field from the instruction. When the value of the two leftmost bits initially contained in Register X1 Right is equal to any of the one or more values specified by the instruction j field, Register Xk shall be cleared in bit positions 1 through 63, and set in bit position 0. When the value of the two leftmost bits initially contained in Register X1 Right is not equal to any of the one or more values specified by the instruction's j field, Register Xk shall be cleared in all 64-bit positions, 0 through 63. The values of the j field and the leftmost two bits initially contained in Register X1 Right shall be interpreted with respect to equality (EQ) as follows:

	Binary Value of Bits 32 and 33 of X1 Right, respectively			
j	00	01	10	11
0 0 0 0	Unconditional inequality			
0 0 0 1				EQ
0 0 1 0			EQ	
0 0 1 1			EQ	EQ
0 1 0 0		EQ		
0 1 0 1		EQ		EQ
0 1 1 0		EQ	EQ	
0 1 1 1		EQ	EQ	EQ
1 0 0 0	EQ			
1 0 0 1	EQ			EQ
1 0 1 0	EQ		EQ	
1 0 1 1	EQ		EQ	EQ
1 1 0 0	EQ	EQ		
1 1 0 1	EQ	EQ		EQ
1 1 1 0	EQ	EQ	EQ	
1 1 1 1	Unconditional equality			



Note: The four individual bits of j can be visualized as individual pointers which are associated, from left to right, with the four possible values (00, 01, 10 and 11) of the tested bit-pair (bits 32 and 33 of Register X1 Right). For example, if j = 0101, equality shall be detected when the value of the tested bit pair is 01 or 11.

2.3 BUSINESS DATA PROCESSING INSTRUCTIONS

The general form of execution for the instructions in this group shall involve the utilization of a first data field in central memory, referred to as the "source," to modify, replace, or compare with a second data field in central memory referred to as the "destination." Both the source and destination fields shall be individually described by means of independently designated Data Descriptors, with respect to the types of representation, sign and zone conventions, lengths and relative locations of the data fields.

The Data Descriptors shall be obtained from central memory at locations immediately following the BDP instruction, as defined by the BDP instruction format and number of descriptors used by the instruction. (See 2.3.1.) All descriptors consist of a 32-bit halfword, aligned to a parcel (16 bit) boundary in central memory.

2.3.1 General Description

The instructions of this group utilize the jk and jkiD instruction formats in combination with one or two descriptors in the following combinations:

(1) jk and two descriptors

	Operation Code	j	k
P	8	4	4

	Descriptor j
P+2	32

	Descriptor k
P+6	32

(2) jkiD and two descriptors

	Operation Code	j	k	i	D
P	8	4	4	4	12

	Descriptor j
P+4	32

	Descriptor k
P+8	32

(3) jkiD and one descriptor

	Operation Code	j	k	i	D
P	8	4	4	4	12

	Descriptor j or k
P+4	32

2.3.1.1 Operation Codes

A total of 18 operation codes shall be utilized by the instructions comprising the BDP instruction group. These instructions are individually listed with their full names in appendix A of this specification. For the purpose of this specification, the BDP Instruction group shall be further divided into three subgroups, including "short" instruction names, as follows:

Note: For the order of exception sensing for these instructions, as well as all other instructions, see paragraph 2.8.7 of this specification.

Subgroup	Short Name
BDP Numeric	Sum Difference Product Quotient Scale Scale Rounded Decimal Compare Numeric Move
Byte	Compare Compare Collated Scan While Non-Member Translate Move Bytes Edit
Immediate Data	Move Immediate Data Compare Immediate Data Add Immediate Data

2.3.1.2 Access Types

For the purpose of establishing operand access validity, every central memory operand access which is performed for the purpose of reading source field data shall be a read type access.

For the purpose of establishing operand access validity, every central memory operand access which is performed for the purpose of writing destination field data shall be a write type access.

For the purpose of establishing operand access validity, every central memory reference operand access which is performed for the purpose of reading data descriptors shall be an execute type access.

2.3.1.3 Undefined Results for Invalid BDP Data

For the execution of any applicable BDP instruction which results in the recording of an Invalid BDP Data condition, if either the corresponding bit in the user mask is clear or traps are disabled, then the results stored into the destination field in central memory shall be undefined for instructions other than Decimal Compare (Op. 74) and Compare Immediate Data (Op. FA), and the results stored in X1 Right (X1R) shall be undefined for both instructions.

2.3.1.4 Overlap

The execution of BDP Instructions shall be undefined with respect to the generated results, for every case in which the source and destination fields overlap and are not coincident in their leftmost and rightmost byte positions.

2.3.2 Data Descriptors

Data Descriptors shall consist of 32-bit halfwords and shall directly follow the BDP instructions referring to them.

A Data Descriptor shall be formatted as follows:

F	D	T	L	0
1	3	4	8	16

00 32-bit Descriptor 31

F = 0, Length = L
F = 1, Length = (X0) for Descriptor associated with Aj
Length = (X1) for Descriptor associated with Ak

The D field is a 3-bit reserved field in bit positions 01, 02 and 03 of the data descriptor. Interpretation of other Data Descriptor fields follows.

2.3.2.1 Data Descriptor Interpretation

For all BDP instructions, the term "D(Aj)" shall be used to denote "the contents of the source data field," addressed by means of the components associated with the BDP instruction's j field designator. Similarly, the term "D(Ak)" shall be used to denote "the contents of the other source field or the destination data field," addressed by means of the components associated with the BDP instruction's k field designator.

2.3.2.1.1 BDP Operand Address, 0 Field

The PVA corresponding to the leftmost byte of a BDP source or destination field shall be obtained by utilizing the 16-bit O field of the corresponding data descriptor (bit positions 16 through 31) as a byte item count to be added as a sign extended 32-bit offset (two's complement for negative offset) to the byte number (BN) portion of the base PVA contained in Register Aj or Ak respectively.

2.3.2.1.2 BDP Operand Type, T Field

The T field shall consist of 4 bits, in bit positions 04 through 07 of the Data Descriptor, and shall describe the type of data representation used in the associated source or destination field. The 12 values of the T field are assigned data type representations as follows:

T	Data Type	Maximum Length (bytes)
0	Packed Decimal No Sign	19
1	Packed Decimal No Sign Leading Slack Digit	
2	Packed Decimal Signed	
3	Packed Decimal Signed Leading Slack Digit	
4	Unpacked Decimal Unsigned	38
5	Unpacked Decimal Trailing Sign Combined Hollerith	
6	Unpacked Decimal Trailing Sign Separate	
7	Unpacked Decimal Leading Sign Combined Hollerith	
8	Unpacked Decimal Leading Sign Separate	
9	Alphanumeric	256
10	Binary Unsigned	8
11	Binary Signed	
12	Reserved	Ignored
13	Reserved	
14	Reserved	
15	Reserved	

As determined by the operation code, source and destination field data types shall be restricted to only those combinations which are defined as valid within the instruction descriptions. The designation of invalid T field combinations within the associated Data Descriptors shall result in the detection of an Instruction Specification Error, the instruction's execution shall be inhibited and the corresponding program interruption shall occur. (See 2.8.1.4.) The term "freely compatible" as used in the BDP instruction descriptions, means that any allowable source field data type may be used with any allowable destination field data type.

2.3.2.1.3 BDP Operand Length, F and L Fields

The length in bytes of a BDP source or destination field shall be obtained according to the value of the 1-bit F field (bit 00) of the corresponding descriptor as follows:

F	Length
0	Obtained from the 8-bit L field (bits 08 through 15) of the corresponding descriptor)
1	Obtained from bits 55 through 63 of X0 Right for the descriptor associated with Aj, and from bits 55 through 63 of X1 Right for the descriptor associated with Ak.

Although field lengths as long as 256 bytes are possible, the length of a BDP operand shall be restricted to a smaller value for decimal and binary operations, according to the operand data type. These inclusive limits are shown in paragraph 2.3.2.1.2.

When any BDP field length exceeds the specified maximum associated with a given data type, an Instruction Specification Error shall be detected, the execution of that instruction shall be inhibited and the corresponding program interruption shall occur. (See 2.8.1.4.)

If F equals 1, then only the rightmost 9 bits of X0 and X1 will be checked to determine whether or not the field length exceeds the maximum allowed. The other bits of X0 and X1 will not be inspected and will be assumed to be all zeros.

2.3.2.2 Data and Sign Conventions

With respect to numeric data and sign conventions, interpretation shall be performed according to Type (T) where applicable, for Characters (C), Digits (D) and Signs (S), using hexadecimal notation, as follows:

Note: Data field examples are illustrated as three byte fields.

a. Type 0: Packed Decimal No Sign

D D	D D	D D

D: Hex (0) through hex (9); Decimal 0 through 9, respectively.

Note: This format corresponds to an even number of digits in the decimal number.

b. Type 1: Packed Decimal No Sign Slack Digit

0 D	D D	D D

0: Hex (0); Decimal 0 (see item q for handling of slack digit).

D: Hex (0) through hex (9); Decimal 0 through 9, respectively.

Note: This format corresponds to an odd number of digits in the decimal number.

c. Type 2: Packed Decimal Signed

D D	D D	D S

D: Hex (0) through hex (9); Decimal 0 through 9, respectively.

S: Hex (A), (B), (C), (E), or (F): positive [hex (C) is preferred];
Hex (D): negative.

Note: This format corresponds to an odd number of digits in the decimal number.

d. Type 3: Packed Decimal Signed Slack Digit

0 D	D D	D S

0: Hex (0); Decimal 0 (see item q for handling of slack digit).

D: Hex (0) through hex (9); Decimal 0 through 9, respectively.

S: Hex (A), (B), (C), (E), or (F): positive [hex (C) is preferred];
Hex (D): negative.

Note: This format corresponds to an even number of digits in the decimal number.

e. Type 4: Unpacked Decimal Unsigned

D	D	D

D: ASCII character 0 through 9 represented by hex (30) through hex (39), respectively.

f. Type 5: Unpacked Decimal Trailing Sign Combined Hollerith

D	D	C

D: ASCII character 0 through 9 represented by hex (30) through hex (39), respectively.

C: An ASCII character decoded as follows:

ASCII 1 through 9 [hex (31) through hex (39)] ASCII <u>A</u> through <u>I</u> [<u>hex (41)</u> through <u>hex (49)</u>]	either represents +1 through +9
ASCII J through R [hex (4A) through hex (4F) and hex (50) through hex (52)]	represents -1 through -9
ASCII <u>{</u>, <, O, & [<u>hex (7B)</u>, hex (3C), hex (30), hex (26)]	represents +0.
ASCII <u>}</u>, !, - [<u>hex (7D)</u>, hex (21), hex (2D)]	represents -0.

Note: The underlined characters and codes are the preferred ones.

g. Type 6: Unpacked Decimal Trailing Sign Separate

D	D	S

D: ASCII character 0 through 9 represented by hex (30) through hex (39), respectively.

S: ASCII character + [hex (2B)]: positive sign;

ASCII character - [hex (2D)]: negative sign.

h. Type 7: Unpacked Decimal Leading Sign Combined Hollerith

C	D	D

C and D have the same meaning as for type 5 in subparagraph f.

i. Type 8: Unpacked Decimal Leading Sign Separate

S	D	D

D and S have the same meaning as for type 6 in subparagraph g.

j. Type 9: Alphanumeric

C	C	C

C: Any ASCII character code.

k. Type 10: Binary Unsigned

The field defined by the number of bytes contains the positive binary value of the operand.

(The unsigned numeric value is always considered to be positive. If negatively signed data is moved to a type 10 receiving field, it too is considered positive.)

l. Type 11: Binary Signed

The field defined by the number of bytes contains the signed binary value of the operand, negative values being represented in the two's complement form.

m. Type 12: Reserved

n. Type 13: Reserved

o. Type 14: Reserved

p. Type 15: Reserved

q. Slack Digit

For data types 1 and 3: The value of the slack digit as read from central memory shall be ignored and treated as the value zero. The value of the slack digit as written into central memory shall be forced to zero, remaining unaffected by any Arithmetic Overflow or Arithmetic Loss of Significance conditions that may occur. (See 2.8.3.10 and 2.8.3.15.)

2.3.3 BDP Numeric

The instructions in this subgroup shall provide the means for performing arithmetic, shift, conversion and comparison operations for byte fields in central memory consisting of numeric decimal data.

Unless the length and type fields with the Data Descriptors associated with the source and destination fields conform to the restrictions defined within the following instruction descriptions, the detection of a Length or Type error shall result in an Instruction Specification Error condition, the execution of the associated instruction shall be inhibited and the corresponding program interruption shall occur.

Overflow into or other alteration of the slack digit of destination field types 1 and 3 is not allowed (see 2.3.2.2, subparagraph q).

The result shall be right justified in the destination field. If the decimal result is shorter than the destination field, the destination field shall be zero filled to the left. If the result is longer than the destination field, the result shall be truncated on the left as necessary. Thus, conceptually, these instructions shall process the data fields from right to left.

Note that these conventions shall cover the end cases for numeric operands of length equal to 1 for all numeric data types. For instance, a Numeric Move (Op. 75) from a type 5 operand to a type 3 or type 6 operand of length 1 would amount to an extraction of the source field sign.

A source BDP operand of numeric type (0 through 8) and a length zero, shall be interpreted as the value zero.

A destination BDP operand of length zero shall transform the associated instruction into a no-op. However, when the source field does not also have a length of zero, exception sensing for the source field shall occur normally (including the testing for Arithmetic Loss of Significance or Arithmetic Overflow condition) with the exception that Divide Fault shall not be detected. When both destination and source fields are of length zero, no data exception testing is performed on either field. (See 2.1.7.)

Minus zero shall be considered equivalent to plus zero by all the instructions in this subgroup, with respect to decimal numeric data. These instructions shall not store minus zero as a result except when truncation of a nonzero, negative field produces a negative zero which will result in negative zero being stored and detection of an Arithmetic Loss of Significance.

The representation for zero, zones and signs shall be normally determined by interpreting the T field from the Data Descriptor associated with the destination field.

Division by zero shall not be allowed to the extent that the destination field in central memory shall not be changed and a Divide Fault condition shall be detected. When the corresponding mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. (See 2.8.3.8.)

Each source digit shall be checked for decimal digit validity. An invalid decimal digit shall cause an Invalid BDP Data condition to be detected. When the corresponding mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. (See 2.8.3.16.)

The sequence of exception sensing for the decimal quotient instruction is as follows:

1) Check D(Aj) for an invalid decimal digit (Invalid BDP Data condition).
2) Check D(Aj) for either zero length or zero value (Divide Fault condition).
3) Check D(Ak) for an invalid decimal digit (Invalid BDP Data condition).

Thus, invalid data in D(Ak) shall result in an Invalid BDP Data condition only in the absence of a Divide Fault condition.

2.3.3.1 Arithmetic

a. Decimal Sum, D(Ak) replaced by D(Ak) plus D(Aj)

70jk (2 descriptors) (Ref. 074)

b. Decimal Difference, D(Ak) replaced by D(Ak) minus D(Aj)

71jk (2 descriptors) (Ref. 075)

c. Decimal Product, D(Ak) replaced by D(Ak) times D(Aj)

72jk (2 descriptors) (Ref. 076)

d. Decimal Quotient, D(Ak) replaced by D(Ak) divided by D(Aj)

73jk (2 descriptors) (Ref. 077)

Operation: These instructions shall arithmetically modify the initial contents of the destination field in central memory, (treated as an augend, minuend, multiplicand or dividend as determined by the operation code) by the contents of the source field in central memory (treated as an addend, subtrahend, multiplier or divisor as determined by the operation code) and shall transfer the decimal result consisting of a sum, difference, product or quotient, as determined by the operation code, to the destination field in central memory.

Divide Fault shall be detected as specified in the following table.

K Field Length	K Value	J Field Length	J Value	Divide Fault
0	*	0	*	No
0	*	Nonzero	0	No
0	*	Nonzero	Nonzero	No
Nonzero	0	0	*	Yes
Nonzero	0	Nonzero	0	Yes
Nonzero	0	Nonzero	Nonzero	No
Nonzero	Nonzero	0	*	Yes
Nonzero	Nonzero	Nonzero	0	Yes
Nonzero	Nonzero	Nonzero	Nonzero	No

* Since field length is zero, the data is not examined.

Types: All Packed decimal types and all Unpacked decimal types, except for the Leading Sign formats, shall be freely allowed for decimal arithmetic; i.e., types 0 through 6, shall be compatible for these instructions.

Unpacked Decimal Leading Sign (both conventions) shall not be supported in the decimal arithmetic. A Numeric Move instruction must be generated to format the operands of those types prior to their use in arithmetic operations.

Lengths: The maximum allowable lengths for the source and destination fields shall be determined according to their respective decimal data types as defined in subparagraph 2.3.2.1.3 of this specification.

Note: Decimal operands shall be treated as integer values.

When the results of these instructions exceed the capacity of the designated destination field such that significant digits are not stored into central memory, an Arithmetic Overflow condition shall be detected. When the corresponding user condition mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.10 of this specification.

The results from these instructions shall be algebraically signed unless they are equal to zero in their entirety and there is no arithmetic overflow, in which case their signs shall be made positive.

These instructions shall generate a result value in accordance with the type T of the destination field and the preferred sign convention for that given type.

2.3.3.2 Shift

a. Decimal Scale, D(Ak) replaced by D(Aj), scaled per XiR plus D

E4jkiD (2 descriptors) (Ref. 078)

b. Decimal Scale Rounded, D(Ak) replaced by rounded D(Aj), scaled per XiR plus D

E5jkiD (2 descriptors) (Ref. 079)

These Shift instructions shall move data initially contained in the source field to the destination field, and shall provide shifting of the data under control of a shift count. The shift count shall be derived in the following manner: the rightmost 8 bits from the instruction's D field shall be added to the rightmost 8 bits initially contained in bit positions 56 through 63 of Register Xi Right and the 8-bit sum shall represent the computed shift count. Any overflow from the 8-bit sum is ignored. In this context, the contents of Register X0 shall be interpreted entirely of zeros. A zero shift count shall cause the instruction to act as a move only instruction.

The 8-bit shift count shall be interpreted as a signed, binary integer. When this 8-bit shift count is positive, the direction of the shift shall be left with the number of decimal digit positions to be shifted determined by the value of the shift count. When this 8-bit shift count is negative, the direction of the shift shall be right with the number of decimal digit positions to be shifted determined by the value of the two's complement of the shift count with 1000 0000 being interpreted as right shift 128. Thus, positive shift counts shall provide the means for multiplying the source data field by powers of ten, and negative shift counts shall provide the means for dividing the source data fields by powers of ten, as the source data is moved to the destination field.

Shift counts shall be interpreted as follows:

```
0111 1111     Left Shift 127
    :              :
    :              :
    :              :
0000 0000     Left Shift 0
----------------------------
1111 1111     Right Shift 1
    :              :
    :              :
    :              :
1000 0001          :
1000 0000     Right Shift 128
```

When nonzero digits are shifted left end-off, or truncated on the left, an Arithmetic Loss of Significance condition shall be detected. If the corresponding user condition mask bit is set, and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. (See 2.8.3.15.)

Shifting shall be accomplished end-off with zero fill on the appropriate end(s) as required to accommodate the length and type of the receiving field. (For example, when the destination field is longer than the source field, and the difference in field lengths is greater than the left shift count, such a scale instruction shall provide zero fill, to the extent required, on both the right and left ends of the destination field result.)

Types: Source field data shall be restricted to types 0 through 6, all of which shall be freely compatible with allowable destination field data types of 0 through 6.

Lengths: The maximum allowable lengths for the source and destination fields shall be determined according to their respective decimal data types as defined in subparagraph 2.3.2.1.3 of this specification.

Operation: These instructions shall move and scale the decimal data field initially contained in the source field to the destination field. They shall transfer the sign of the source field to the destination field without change, (unless the results consist entirely of zeros and there is no loss of significance, in which case the sign of the destination field shall be made positive or unless the result would otherwise contain a nonpreferred sign in which case the sign of the destination field shall contain the preferred sign).

Scale Rounded: When specified by means of the operation code, rounding shall be performed for negatively signed scale factors by adding five to the last digit shifted end-off and propagating carries, if any, through the decimal result transferred to the destination field. Thus, the absolute value shall be rounded upwards.

2.3.3.3 Move

Numeric Move, D(Ak) replaced by D(Aj) after formatting

75jk (2 descriptors) (Ref. 092)

This instruction shall format the number obtained from the source field and shall transfer the result to the destination field.

The source field shall be validated according to the T field from its associated descriptor; the source field shall be reformatted according to the T field from the data descriptor associated with the destination field and the result shall be transferred to the destination field.

The format of the different data types allowed in this instruction are described in subparagraph 2.3.2.2 of this document. The conversion and format operation shall be performed on any combination of fields of type 0 through 8 or 10 or 11.

If the source has a decimal data type and the destination a binary data type, a conversion from decimal to binary shall be performed. In this case, the maximum length for the source shall be determined by the decimal data type: 19 bytes for types 0 through 3, and 38 bytes for types 4 through 8; the maximum field length for the destination shall be 8 bytes. If the destination field is not long enough to accommodate the entire binary number, truncation of the leftmost bytes shall occur. If the destination field is longer than the result of the conversion, the sign bit shall be extended on the left.

If the source has a binary data type and the destination a decimal data type, a conversion from binary to decimal shall be performed. The length restrictions on the operands are the same as in the previous case. If the destination field is too short to accommodate the converted number, leading digits shall be truncated according to the destination's data type. If the receiving field is longer than the converted number, leading zeros shall be supplied in accordance with the decimal data type: ASCII character zero [hex(30)] or digit zero [hex(0)].

When truncation of data results in loss of significance, an Arithmetic Loss of Significance condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, program interruption shall occur. Execution of this instruction (specifically storing into central memory) may or may not be inhibited as determined on a model-dependent basis. However, when program interruption occurs, the PVA in the P Register recorded in either the exchange package or the stack frame save area shall point to this instruction.

When both operands are decimal, their maximum allowable lengths shall be determined according to their respective decimal data types as defined in subparagraph 2.3.2.1.3, of this specification.

When both operands are decimal, the destination shall be filled from right to left. Unequal field lengths shall result either in truncation of the leading digits or in insertion of leading zeros according to the destination data type: ASCII character zero [hex(30)] or digit zero [hex(0)], for unpacked and packed decimal data types, respectively.

2.3.3.4 Comparison

Decimal Compare, D(Aj) to D(Ak), result to X1R

74jk (2 descriptors) (Ref. 083)

This instruction shall algebraically compare the decimal contents of the source field to the decimal contents of the destination field and shall transfer a 32-bit halfword to Register X1 Right according to the results of the comparison.

When the contents of the source and destination fields are equal, the entire 32-bit positions of Register X1 Right shall be cleared.

When the contents of the source field are greater than the contents of the destination field, Register X1 Right shall be cleared in bit position 32 and 34 through 63, and shall be set in bit position 33.

When the contents of the source field are less than the contents of the destination field, Register X1 Right shall be cleared in bit positions 34 through 63 and shall be set in bit positions 32 and 33.

Types: All Packed decimal types and all Unpacked decimal data types except for the Leading Sign formats, shall be freely allowed in comparisons; i.e., types 0 through 6, shall be compatible for this instruction.

Lengths: Lengths shall be confined to the same maximum values as for a Decimal Difference instruction. Unequal field lengths shall be accommodated by providing zero fill in the leftmost positions, as required, for the field having the shorter length. The maximum number of bytes occupied by each operand is a function of its data type and is specified in subparagraph 2.3.2.1.3, of this specification.

2.3.4 Byte

The instructions in this subgroup shall provide the means for comparing, scanning, translating, moving, and editing byte fields in central memory to the extent defined by the following instruction descriptions.

These instructions shall utilize spaces for extending alphanumeric (type 9) fields, with the space being represented by hex(20).

A source byte operand of length zero shall be functionally interpreted as a string of space characters (ASCII character: hex(20)) for all the instructions in this subgroup except "Edit."

Decimal Significance Loss shall not be detected for the instructions in this subgroup.

2.3.4.1 Comparison

a. Byte Compare, D(Aj) to D(Ak), result to X1R, index to X0R

77jk (2 descriptors) (Ref. 084)

b. Byte Compare Collated, D(Aj) to D(Ak), both translated per (Ai plus D), result to X1R, index to XOR

E9jkiD (2 descriptors) (Ref. 085)

These instructions shall compare the bytes contained in the source field to the bytes contained in the destination field and shall transfer the results of that comparison to Register X1 Right.

The comparison shall proceed from left to right. When the field lengths are unequal, trailing space characters shall be used for the field having the shorter length. The maximum length for each operand shall be 256 bytes.

These instructions shall ignore the type field. Each byte from the source and destination field shall be treated as an 8-bit quantity having an absolute value with respect to the comparison operation.

The comparison shall continue until the longer field has been exhausted or until an "inequality" is detected between corresponding bytes from the source and destination fields according to the following definitions. For the Compare instruction, inequality between the bytes obtained directly from the source and destination fields shall result in the completion of the comparison. For the Collated Compare instruction inequality of the bytes obtained directly from the source and destination fields shall result in the translation of both bytes, by means of a translation table, and inequality of the post-translation bytes shall result in the completion of the comparison. When the translated bytes are equal, and the longer field has not been exhausted, comparison between the corresponding bytes obtained directly from the source and destination fields shall be resumed.

When every byte associated with the source field is equal to every corresponding byte associated with the destination field, (including the trailing space characters if any), the entire 32-bit positions of Register X1 Right shall be cleared. When the first inequality between bytes occurs as a result of a byte associated with the source field having a greater value than the corresponding byte associated with the destination field, Register X1 Right shall be cleared in bit positions 32 and 34 through 63, and shall be set in bit position 33. When the first inequality between bytes occurs as a result of a byte associated with the source field having a value less than the corresponding byte associated with the destination field, Register X1 Right shall be cleared in bit positions 34 through 63 and shall be set in bit positions 32 and 33. In addition, the sequence number of the byte which caused the first inequality will be placed in Register X0 Right. (Note: the sequence number shall be initialized to zero. Moreover, when one of these instructions terminates as a result of inequality, the value of the sequence number transferred to Register X0 Right, if added to the leftmost byte addresses of the source and destination fields, will provide the addresses of the source and destination field bytes, respectively, which caused the inequality.) If no inequalities are found, Register X0 Right shall remain unchanged.

Translation table: The translation table used for each occurrence of direct inequality during Collated Compare instructions, shall be addressed by a PVA whose Ring Number (RN) and Segment (SEG) are obtained from Ai, and whose Byte Number (BN) is formed by the 32-bit sum (ignoring overflow) of the rightmost 32 bits of Ai plus the instruction's 12-bit D field extended to the left with 20 zeros. The entire table, consisting of 256 bytes, may be loaded internally to the processor, on a model dependent basis before any operation on the data is performed.

Each byte shall be translated by using its value as a positive offset to be added to the beginning (leftmost) address of the Translation Table, (Ai) + D, for the purpose of addressing the translated byte to be read from central memory.

Invalid Segment/Access Violation: The associated program interruption shall occur as described in paragraph 2.8.1 and the execution of this instruction shall be inhibited when either an Inalid Segment or an Access Violation is recorded.

Address Specification Error/Page Table Search Without Find: The associated program interruption shall occur as described in paragraph 2.8.1 and the execution of this instruction shall be inhibited when either an Address Specification Error or a Page Table Search Without Find is recorded except when the exception is associated with unneeded data. In this case the processor may but need not report the exception condition. Not reporting one of the two exceptions can only occur when a source field spans two pages and the exception is associated with the second page and a mismatch occurs using the data from the first page.

2.3.4.2 Byte Scan

Byte Scan While Nonmember, D(Ak) for presence bit in (Ai plus D), character to X1R, index to X0R

F3jkiD (1 descriptor) (Ref. 086)

Operation: The operation shall proceed from left to right on the destination field addressed by D(Ak). One character at a time shall be taken from this character string and used as a bit address into the string addressed by a PVA whose Ring Number (RN) and Segment (SEG) are obtained from Ai, and whose Byte Number (BN) is formed by the 32-bit sum (ignoring overflow) of the rightmost 32 bits of Ai plus the instruction's 12-bit D field extended to the left with 20 zeros. The scan shall terminate if the bit thus addressed is ON or if the destination field has been exhausted; otherwise the next character in D(Ak) is considered.

Source Field: The operand addressed by Ai+D shall be interpreted as a bit string consisting of 256 bits (32 bytes). The entire table, consisting of 256 bits, may be loaded internally to the processor, on a model dependent basis, before any operation on the data is performed.

Destination Field: The type field in D(Ak) shall be ignored. The operand addressed by D(Ak) shall be interpreted as a byte string, and restricted to no more than 256 characters.

The binary value of the sequence number in the string, of the byte which caused the scan to terminate shall be placed right-justified into X0 Right.

The binary value of the character itself which caused the scan to terminate shall be placed right-justified into X1 Right.

If the scan stops by exhaustion of the characters in the byte string, X0 Right shall contain the length of the original byte string and X1 Right shall be set in bit position 32 and cleared in bit positions 33 through 63.

Note: The function Byte Scan While Member can be performed by means of the Byte Scan While Non-Member if the bit string specifying the characters not allowed in the byte string has been previously logically negated.

2.3.4.3 Translate

Byte Translate, D(Ak) replaced by D(Aj), translated per (Ai plus D)

EBjkiD (2 descriptors) (Ref. 088)

This instruction shall translate each byte contained in the source field, according to the translation table in central memory and shall transfer the results of the byte-by-byte translation to the destination field.

The translation table shall be addressed in a manner identical to that previously described for the Byte Compare Collated instruction in subparagraph 2.3.4.1 of this specification. The Type fields in the Data Descriptors associated with the source field and the destination field shall be ignored. Both operands shall be restricted to no more than 256 bytes.

The translation operation shall occur from left to right with each source byte used as a positive offset to be added to the beginning (leftmost byte) address of the translation table for the purpose of permitting each byte's translation. Translated bytes, thus obtained from the translation table, shall be transferred to the destination field. The translation operation shall terminate after the destination field length has been exhausted. When the source field length is greater than the destination field length, rightmost bytes from the source field shall be truncated, to the extent required, with respect to the translation operation. When the source field length is less than the destination field length, translated space characters shall be used to fill the rightmost byte positions of the destination field to the extent required.

2.3.4.4 Move

Move Bytes, D(Ak) replaced by D(Aj)

76jk (2 descriptors) (Ref. 089)

This instruction shall move the bytes contained in the source field to the destination field. The operation shall be performed from left to right with unequal field lengths accommodated by the truncation of trailing characters from the source field or the insertion of trailing spaces into the destination field. The type fields of the source and destination data descriptors shall be ignored. Field lengths shall be restricted to a maximum of 256 bytes.

Invalid Segment/Access Violation: The associated program interruption shall occur as described in paragraph 2.8.1 and the execution of this instruction shall be inhibited when either an Inalid Segment or an Access Violation is recorded.

Address Specification Error/Page Table Search Without Find: The associated program interruption shall occur as described in paragraph 2.8.1 and the execution of this instruction shall be inhibited (except that some portion of the destination field may have meen modified) when either an Address Specification Error or a Page Table Search Without Find.

2.3.4.5 Edit

Edit, D(Ak) replaced by D(Aj) edited per (Ai plus D)

EDjkiD (2 descriptors) (Ref. 091)

This instruction shall edit the digits or characters contained in the source field according to an edit mask in central memory and shall transfer the result to the destination field. The edit mask shall be addressed by a PVA whose Ring Number (RN) and Segment (SEG) are obtained from Ai, and whose Byte Number (BN) is formed by the 32-bit sum (ignoring overflow) of the rightmost 32-bit of Ai plus the instruction's 12-bit D field extended to the left with 20 zeros. The edit mask shall consist of a one byte length indication followed by a string of micro-operations. The length indication shall include the byte containing the length. (Also see appendixes C & H.)

The edit instruction shall terminate as a result of exhausting the edit mask or under control of the edit mask, i.e., MOP15 with the zero flag FALSE. For both of these circumstances, no exception conditions shall be associated with the completion of the edit instruction even though the source and the destination fields may not have been exhausted. In the event that the destination field is not filled, the remaining portion of the destination field shall not be altered. In the event that the source field is not exhausted, the entire source field shall be checked for invalid BDP data and the sign examined. However, when the interpretation of the edit mask would otherwise result in reading beyond the end of the source field or would result in writing beyond the end of the destination field, an Invalid BDP Data condition shall be detected. Thus a destination field length of zero allows the Edit instruction to proceed until the first output is produced at which point an Invalid BDP Data condition

shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See paragraph 2.8.3.16 of this specification.

Type: The Source Data Descriptor type field shall be confined to the following types: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. A type 9 source field is assumed zero (ZF, zero field = true) and positive. The Destination Data Descriptor type field shall be ignored, and the output field formatted as per type 9. An Instruction Specification Error shall be detected when the source data type is 10, 11, 14, or 15. This condition shall be detected even if the edit mask has a length of 0 or 1.

Special Conventions: The edit operation shall utilize the tables and toggles listed below.

a. Special Characters Table (SCT): The SCT is an eight byte table that shall be initialized by the machine at the start of each edit operation to contain the following:

```
blank fill character
↓    suppression character
↓    ↓    positive sign
↓    ↓    ↓    negative sign
↓    ↓    ↓    ↓
↓    ↓    ↓    ↓
```

Table Index	0	1	2	3	4	5	6	7
Character	b	b	+	-	,	.	$	/
Hexadecimal Value	2D	20	2B	2D	2C	2E	24	2F

Entries in the SCT shall be readable/writable under control of certain micro-operations comprising the mask.

b. The Symbol (SM): The symbol is a string of 0 to 15 characters that shall be created under control of the edit mask and inserted into the destination field under control of the edit mask. Once the SM has been inserted into the destination field, it must be recreated before it can be inserted again. At the start of an edit operation, the SM shall have a zero length.

The SM shall be utilized for the floating sign and floating currency editing features. It shall also be utilized for sign sensitive and significance sensitive character string insertion.

c. End Suppression Toggle (ES): This toggle controls zero suppression. At start of edit, the ES shall be initialized FALSE. The ES shall be set TRUE when zero suppression ends.

d. Negative Sign Toggle (SN): This toggle signifies the sign of the source field. At start of edit, the SN shall be initialized FALSE for an unsigned numeric or a positive numeric source field. It shall be initialized TRUE for a negative numeric source field, only.

e. Zero Field Toggle (ZF): This toggle signifies whether the source field is zero or nonzero. It shall be initialized TRUE.

Source Field Sign: For separately signed numeric data types, the bit positions in the source field which are occupied by the sign shall be automatically skipped with respect to source field addressing under control of the edit mask. For combined sign data types, only the numeric value shall be interpreted with respect to read references of the source field sign byte position under control of the edit mask.

Edit Micro-Operations: The mask shall be interpreted as a string of one byte micro-instructions with the following format.

0 3	4 7
MOP	SV

The MOP is a micro-operator (MOP). It specifies an editing function. The SV is a specification value (SV). Its meaning varies according to the specific MOP which it follows.

Edit control shall proceed from left to right on the mask, one character (or micro-operation) at a time. After interpretation of the micro-operation, action shall be taken on the source and destination field characters (or source digits) which shall also be operated from left to right.

Indexing through the source field shall be by bytes unless its data-type is packed numeric when indexing shall be by half-bytes. Indexing through the destination field shall be by bytes.

Notation for MOP descriptions.

ES	End suppression toggle.
SCT	Special character table.
SCT(n)	(n+1)th entry in the SCT (n must be 0-7).
SV	Specification value.

Note: The one byte length indication contained in the leftmost byte position of the Edit Mask shall include itself in specifying the length of the Edit Mask. (Thus, a maximum of 254 micro-operations may be specified by this byte.)

When the value of the leftmost byte of the Edit Mask is equal to zero or one, the associated Edit instruction shall result in no operation; however, the entire input field (except when type 9) is checked for valid data.

Although not included in each description, prior to the execution of each micro-operation the edit mask index shall be incremented by one.

Micro-operations - MOPs

MOP 0

1. End MOP if SV=0.
2. Set ES true.
3. Translate SV (1 to 15) digits from the source field to their equivalent ASCII characters and copy them into the destination field. The source field must not be type 9 or an Invalid BDP Data condition will be detected. When the corresponding user mask bit is set and traps enabled, instruction execution shall be inhibited and program interruption shall occur.
4. Perform the translate as specified by the NUMERIC function.

MOP 1

1. End MOP if SV=0.
2. Set ES true.
3. Move SV (1 to 15) characters from the source field to the destination field. The source field must be type 9 or an Invalid BDP Data condition will be detected. When the corresponding user mask bit is set and traps enabled, instruction execution shall be inhibited and program interruption shall occur.

MOP 2, 3

No operation.

MOP 4

1. End MOP if SV=0.
2. Move SV (1 to 15) characters from the edit mask to the destination field. When execution of this MOP would require reading beyond the end of the edit mask, an Invalid BDP Data condition shall be detected. When the corresponding user mask bit is set and traps enabled, instruction execution shall be inhibited and program interruption shall occur.

MOP 5

Set the symbol to a single character representing the sign of the source data field.

- Negative source data field

  Copy SCT_3 to the symbol field.

- Positive source data field

  Copy the character (SCT_{SV}) selected from the SCT indexed by the rightmost 3 bits of SV into the symbol field.

MOP 6

1. End MOP if SV=0.
2. Move SV (1 to 15) characters from the edit mask to the symbol. When execution of this MOP would require reading beyond the end of the edit mask, an Invalid BDP Data condition shall be detected. When the corresponding user mask bit is set and traps enabled, instruction execution shall be inhibited and program interruption shall occur.

MOP 7

1. End MOP if SV=0.
2. Translate SV (1 to 15) digits from the source field to their equivalent ASCII characters and copy them into the destination field. The source field must not be type 9 or an Invalid BDP Data condition will be detected. When the corresponding user mask bit is set and traps enabled, instruction execution shall be inhibited and program interruption shall occur.

- ES False AND zero digit

  Copy SCT_1 to the destination field.
- ES False AND nonzero digit

  Set ES true and copy the symbol to the destination field followed by the translated digit.
- ES True

  Copy the translated digit to the destination field.

MOP 8

- ES True

  No operation.
- ES False

  Set ES true and copy the symbol to the destination field.

MOP 9

- SV > 7

  Copy the Symbol to the destination field.
- SV ≤ 7

  Copy the character (SCT_{SV}) selected from the SCT indexed by the rightmost 3 bits of SV into the destination field.

MOP A

- SV > 7
  - Source field positive

    Copy the Symbol to the destination field.
  - Source field negative

    Copy the SCT_0 character to the destination field once for each character in the Symbol.
- SV ≤ 7
  - Source field positive

    Copy the character (SCT_{SV}) selected from the SCT indexed by the rightmost 3 bits of SV into the destination field.
  - Source field negative

    Copy SCT_0 into the destination field.

MOP B

This MOP is identical to MOP A with the action caused by the source field sign being exactly reversed.

MOP C

- SV > 7
  - ES True

    Copy the Symbol to the destination field.
  - ES False

    Copy the SCT_1 character to the destination field once for each character in the Symbol.
- SV ≤ 7
  - ES True

    Copy the character (SCT_{SV}) selected from the SCT indexed by the rightmost 3 bits of SV into the destination field.
  - ES False

    Copy SCT_1 into the destination field.

MOP D

Copy the next character from the edit mask into the SCT as indexed by the rightmost 3 bits of SV.

When execution of this MOP would require reading beyond the end of the edit mask, an Invalid BDP Data condition shall be detected. When the corresponding user mask bit is set and traps enabled, instruction execution shall be inhibited and program interruption shall occur.

MOP E

1. End MOP if SV = 0.
2. Copy SCT_1 into the destination field SV (1 to 15) times.

MOP F

1. End MOP if SV = 0.
2. If ZF False (Nonzero Source Field)

   Terminate the Edit instruction.
3. If ZF True (Zero Source Field)

   Reset to start of Destination field and copy SCT_1 into the destination field SV times. Execution of this reset causes all characters previously transmitted to the destination field to be, in effect, discarded (even when more than SV characters were previously transmitted).

Function NUMERIC

This function shall be used by micro-operations 0 and 7 to move a source digit into the destination field.

Each source digit shall be checked; invalid decimal digits shall cause an Invalid BDP Data condition to be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See paragraph 2.8.3.16 of this specification.

When the source field is packed numeric, appropriate ASCII zone bits shall be supplied for the destination character.

A nonzero digit shall cause the ZF toggle to be set FALSE.

2.3.5 Calculate Subscript

This instruction (Op. F4) has been deleted from the CYBER 180 instruction set.

2.3.6 Immediate Data

Within this instruction group, the Immediate Data Byte is an 8-bit field formed by the two's complement addition of bits 56 - 63 (Xi) Right and the rightmost 8 bits of the instruction's D field. Overflow is ignored on this summation. In this context, the contents of Register X0 shall be interpreted as consisting entirely of zeros.

2.3.6.1 Move Immediate Data, D(Ak) replaced by XiR plus D per j

F9jkiD (1 descriptor) (Ref. 154)

This instruction shall move the Immediate Data Byte to the destination field after format conversion per the destination field type and the j field suboperation code. The Immediate Data Byte is described in paragraph 2.3.6. The least significant 2 bits of the j field shall be used as an encoding of the operation to be performed:

a. If = 00, the unsigned (considered positive) numeric value (type 10) contained in the Immediate Data Byte shall be moved right-justified to the receiving field, which must be of type 10 or 11. If necessary, the destination field is filled with zeros on the left.

b. If = 01, the decimal numeric value (type 4) contained in the Immediate Data Byte shall be moved right-justified, to the receiving field after possible reformatting to match the data type of the destination. If the format requires a sign, a positive sign shall be supplied. The destination shall be restricted to one of the decimal data types 0 through 6. This move shall be executed according to the rules of the numeric move for truncation, padding and validation.

   Each source digit shall be checked for decimal digit validity. An invalid decimal digit shall cause an Invalid BDP Data condition to be detected. When the corresponding user mask bit is set, and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur.

   This operation will not alter the slack digit of destination field types 1 and 3 (see 2.3.2.2, subparagraph q). When truncation of numeric data results in loss of significance, an Arithmetic Loss of Significance condition shall be detected. If the corresponding user condition mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. (See 2.8.3.15.)

c. If = 10, the ASCII character contained in the Immediate Data Byte is repeated left to right in the receiving field. The destination data type shall be ignored.

d. If = 11, the ASCII character contained in the Immediate Data Byte is moved left-justified into the receiving field, the rest of that field is space filled. The destination data type shall be ignored.

2.3.6.2 Compare Immediate Data, XiR plus D to D(Ak) per j, result to X1R

FAjkiD (1 descriptor) (Ref. 155)

This command shall, depending on the value of the j field, compare the explicit value contained in the Immediate Data Byte to D(Ak) after a possible reformatting to match the data type and shall transfer a 32-bit halfword to Register X1 Right according to the result of the comparison. The Immediate Data Byte is described in paragraph 2.3.6.

When the contents of the source and destination fields are equal, the entire 32-bit positions of Register X1 Right shall be cleared.

The rightmost two bits of the j field shall be used as an encoding of the operation to be performed:

a. If j=00, the unsigned (considered positive) numeric value (type 10) contained in the Immediate Data Byte shall be compared to the contents of field D(Ak), which must be of type 10 or 11. If field D(Ak) is longer than one byte, then the Immediate Data Byte will be zero-filled to the left as necessary.

b. If j=01, the decimal numeric value (type 4) contained in the Immediate Data Byte shall be compared to the contents of field D(Ak) after possible reformatting to match the data type of field D(Ak). If the format requires a sign, a positive sign shall be supplied. The D(Ak) field shall be restricted to one of the decimal data types 0 through 6. If field D(Ak) is longer than one byte, then the Immediate Data Byte shall be zero-filled to the left as necessary.

   Each source digit shall be checked for decimal digit validity. An invalid decimal digit shall cause an Invalid BDP Data condition to be detected. When the corresponding user mask bit is set, and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur.

c. If j=10, the ASCII character contained in the Immediate Data Byte shall be compared left to right with each successive byte contained in the D(Ak) field. The data type of field D(Ak) shall be ignored.

d. If j=11, the ASCII character contained in the Immediate Data Byte shall be compared to the leftmost byte in field D(Ak). If the comparison is equal and if field D(Ak) is longer than one byte, then a space character shall be compared left to right with each successive remaining byte contained in the D(Ak) field. The data type of field D(Ak) shall be ignored.

When the contents of the source field are greater than the contents of the destination field, Register X1 Right shall be cleared in bit positions 32 and 34 through 63, and shall be set in bit position 33.

When the contents of the source field are less than the contents of the destination field, Register X1 Right shall be cleared in bit positions 34 through 63 and shall be set in bit positions 32 and 33.

The interpretation of the source and destination fields are analogous to those described under the Move Immediate Data instruction, paragraph 2.3.6.1.

2.3.6.3 Add Immediate Data, D(Ak) replaced by D(Ak) plus XiR plus D per j

FBjkiD (1 descriptor) (Ref. 156)

Operation: This command shall add the explicit integer value contained in the Immediate Data Byte to D(Ak) after a possible conversion to match the destination data type. The Immediate Data Byte is formed as described in 2.3.6.

Source: The Immediate Data Byte is used to store the integer value of the addend. The j field is used as an encoding of the type of the data contained in the Immediate Data Byte. The least significant bit of the j field is decoded as follows:

a. If = 0, the Immediate Data Byte contains an unsigned (considered positive) binary integer value; Immediate Data Byte = Data Type 10.

b. If = 1, the Immediate Data Byte contains one ASCII character representing a decimal digit; if invalid decimal data is encountered in the Immediate Data Byte, an Invalid BDP Data condition shall be detected. When the corresponding user mask bit is set, and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. Immediate Data Byte = Data Type 4.

If the source corresponds to case a. above, the destination shall be confined to types 10 and 11.

If the source corresponds to case b. above, the destination shall be confined to types 0 through 6.

If unauthorized data types are specified, an Instruction Specification error shall be detected, the instruction's execution shall be inhibited, and the corresponding program interruption shall occur. (See 2.8.1.4.)

Overflow into the slack digit of destination field types 1 and 3 is not allowed (see 2.3.2.2, subparagraph q). When the results of the add operation exceed the capacity of the destination field, an Arithmetic Overflow condition shall be detected. If the corresponding user condition mask bit is set and the trap is enabled, instruction execution shall be inhibited and the program interruption shall occur. (See 2.8.3.10.)

2.4 FLOATING POINT INSTRUCTIONS

A floating point number shall consist of a signed exponent and a signed fraction. The signed fraction shall also be referred to as the coefficient.

The quantity expressed by a floating point number shall be of the form $(f)2^x$ where f represents the signed fraction and x represents the signed exponent of the base 2.

The exponent base of 2 shall be an implied constant for all floating point numbers and thus shall not explicitly appear in any floating point format.

2.4.1 Format: 64-Bit

All 64-bit floating point data shall have one of two fixed length formats: a 64-bit word called single precision, or a 128-bit word called double precision.

In both the single and double precision formats, the leftmost bit position, 00, shall be occupied by the sign of the fraction. The fifteen bit positions immediately to the right of bit 00, 01 through 15, shall be occupied by the signed exponent.

The field immediately to the right of the signed exponent shall be occupied by the fraction which in single precision format shall consist of 48 bits and in double precision format shall consist of 96 bits, according to the following figures.

00	01 15	16 63
S	Signed Exponent	48-bit Fraction

Single Precision Floating Point Number

00	01 15	16 63
S	Signed Exponent	Leftmost 48 bits of the 96-bit Fraction

64	65 79	80 127
S	Signed Exponent	Rightmost 48 bits of the 96-bit Fraction

Double Precision Floating Point Number

A double precision floating point number shall consist of two single precision floating point numbers located in consecutively numbered X Registers. The two single precision floating point numbers comprising a double precision floating point number shall be referred to as the leftmost and rightmost parts as contained in the Xn and Xn+1 Registers, respectively. The leftmost part may be any single precision floating point number and when it is normalized, (the leftmost bit of the fraction, in bit position 16, is equal to a one) the double precision floating point number shall be considered to be normalized. The sign of the fraction and the exponent of the leftmost part shall constitute the sign of the fraction and the exponent of the double precision number.

The fraction field of the leftmost part shall constitute the leftmost 48 bits of the 96-bit double precision fraction. The fraction field of the rightmost part shall constitute the rightmost 48 bits of the 96-bit double precision fraction. The sign of the fraction and the exponent of the rightmost part shall not be utilized from any number constituting an input operand (argument) to a double precision floating point operation. However, the formation of a double precision floating point result shall include making the sign of the fraction of the rightmost part the same as that of the leftmost part and shall also include making the exponent of the rightmost part equal to the exponent of the leftmost part.

2.4.1.1 Standard Numbers

The fraction field of a floating point number shall have its binary point immediately to the left of its leftmost bit position, 16. Both positive and negative quantities shall have a true fraction with the sign indicated solely by means of the sign bit. A number shall be positive or negative depending on whether the sign is a zero or a one, respectively.

```
   |<------- Exponent ------->|<------------ True Fraction ------------>|
 +--+--+------+---------------+-----------------------------------------+
 |00|01|02  03|04           15|16                                     63|
 +--+--+------+---------------+-----------------------------------------+
  ↑  ↑  ↑  ↑                   ↑
  ↑  ↑  ↑  ↑                   Binary Point
  ↑  ↑  Exponent Out of Range (02, 03)
  ↑  Exponent Bias
 Sign of the Fraction
```

The fraction shall be considered to be multiplied by the power of 2 expressed by the exponent which, in encoded form, occupies bit positions 01 through 15. The exponent field shall be used to represent both standard and nonstandard floating point numbers. Standard floating point numbers shall have an actual exponent range from −4096 to +4095 inclusive, and shall be encoded into the exponent field by adding a bias equal to 2^{14}. The effect of biasing the exponent is demonstrated in table 2.4-1 for standard floating point numbers in which the ascending order from smallest to largest encoded representations corresponds to the smallest to largest progression of multiplier values represented by the actual exponents in the range of −4096 to +4095 inclusive.

The ranges in magnitude, M, covered by standard, normalized floating point numbers in each of the two formats is as follows:

> Single precision; $2^{-4097} \leq M \leq (1-2^{-48})2^{4095}$
> (Approximately 14.4 decimal digits of precision)
>
> Double precision; $2^{-4097} \leq M \leq (1-2^{-96})2^{4095}$
> (Approximately 28.9 decimal digits of precision)

For both formats these ranges approximate to:

> $(4.8)\ 10^{-1234} \leq M \leq (5.2)\ 10^{1232}$

2.4.1.1.1 Z3

As shown in table 2.4-1, +Z3 and −Z3 are standard floating point numbers with zero coefficients. The existance of −Z3 in the floating point number set plus the interpretation of +Z3 greater than −Z3 by the floating point compare means that special consideration must be given to −Z3. Add and subtract are the only floating point operations which can produce a Z3 result from normalized input operands and will force any −Z3 result to a +Z3. Multiply and divide operations will only produce Z3 for unnormalized input operands and will produce either +Z3 or −Z3 depending on the signs of the input operands. This −Z3 gives rise to some anomalies. For example, when A = −Z3, and B and C are both nonzero standard numbers in the following equation:

$$A\,(B + C) = AB + AC$$

the comparison reduces to

$$-Z3 \neq +Z3$$

This is true because the floating point compare rules described in 2.4.5 interpret operands with different signs to be unequal. These anomalies only occur when unnormalized operands are used.

2.4.1.1.2 N

As shown in table 2.4-1, +N and −N are those standard floating point numbers which have nonzero coefficients.

2.4.1.2 Nonstandard Numbers

The exponent field shall also be used to represent nonstandard floating point numbers referred to as Zero (±Z1, ±Z2), Infinity (±INF), and Indefinite (±INDEF).

Table 2.4-1 illustrates hexadecimal exponent codes for corresponding nonstandard as well as standard floating point numbers.

2.4.1.2.1 ±Z1, ±Z2

Nonstandard floating point numbers constituting input arguments to floating point operations shall be treated as if they consisted entirely of zeros when bits 01 and 02 are equal to zeros and also when bits 01 and 03 are equal to zeros.

The nonstandard zero floating point numbers are represented as ±Z1 or ±Z2 as shown in table 2.4-1. The specific number in the +Z1 range which consists of all (64) zeros is termed +0. Thus, wherever +Z1 is indicated, the +0 is also included since it is a member of ±Z1.

2.4.1.2.2 ±INF

Nonstandard floating point numbers constituting input arguments to floating point operations shall be treated as infinite values when bit 01 is equal to one and bits 02 and 03 are not equal to each other.

The nonstandard floating point numbers in the Infinite range are represented as ±INF as shown in table 2.4-1. The specific number in the +INF range which consists of 500....000 is termed +∞. The specific number in the −INF range which consists of D00....000 is termed −∞. Thus wherever ±INF is indicated, the numbers ±∞ are also included since they are members of ±INF.

2.4.1.2.3 ±INDEF

Nonstandard floating point numbers constituting input arguments to floating point operations shall be treated as indefinite values when bits 01 through 03 are all equal to ones.

The nonstandard floating point numbers in the Indefinite range are represented as ±INDEF as shown in table 2.4-1. The specific number in the +INDEF range which consists of 700....000 is termed +IND. Thus, wherever ±INDEF is indicated, the number +IND is also included since it is a member of ±INDEF.

	Hexadecimal exponent including the coefficient sign	Actual exponent (to the base 2)	Standard or nonstandard numbers	Number Classification	Comments	
Positive Numbers (Coeff'nt sign = 0)	7XXX	----	Nonstd.	Indefinite	+INDEF	The specific number 7000...00 is termed +IND
	6FFF : 5000	$2^{12,287}$: $2^{4,096}$		Infinite	+INF	The specific number 5000...00 is termed +∞
	4FFF : 4000	$2^{4,095}$: 2^{0}	Standard	Standard	+N	Standard numbers with a nonzero coefficient
	3FFF : 3000	2^{-1} : $2^{-4,096}$			+Z3	Standard numbers with a zero coefficient
	2FFF : 1000	$2^{-4,097}$: $2^{-12,288}$	Nonstd.	Zero	+Z2	
	0XXX	$2^{-12,289}$: $2^{-16,384}$			+Z1	The specific number 0000...00 is termed +0
Negative Numbers (Coeff'nt sign = 1)	8XXX	$2^{-16,384}$: $2^{-12,289}$			-Z1	
	9000 : AFFF	$2^{-12,288}$: $2^{-4,097}$			-Z2	
	B000 : BFFF	$2^{-4,096}$: 2^{-1}	Standard	Standard	-Z3	Standard numbers with a zero coefficient
	C000 : CFFF	2^{0} : $2^{4,095}$			-N	Standard numbers with a nonzero coefficient
	D000 : EFFF	$2^{4,096}$: $2^{12,287}$	Nonstd.	Infinite	-INF	The specific number D000...00 is termed -∞
	FXXX	----		Indefinite	-INDEF	

Table 2.4-1. 64-Bit Floating Point Representation

2.4.1.3 Exponent Arithmetic

When the exponent fields from input arguments are added, as for floating point multiplication, or subtracted, as for floating point division, the exponent arithmetic shall be performed algebraically in two's complement mode. Moreover, such operations shall take place, conceptually, as if the bias were removed from each exponent field prior to performing the addition or subtraction and then restored following exponent arithmetic so as to correctly bias the exponent result.

Exponent Underflow and Overflow conditions shall be detected for all single precision, but only for the leftmost part of double precision floating point results.

2.4.1.4 Normalization

A normalized floating point number shall have a one in the leftmost bit position, 16, of the fraction field. If the leftmost bit of the fraction is a zero, the number shall be considered unnormalized. Normalization shall take place when intermediate results are changed to final results. Numbers with zero fractions cannot be normalized and such fractions shall remain equal to zero.

For intermediate results in which coefficient overflow has not occurred and the initial operands were normalized, the normalization process shall consist of left shifting the fraction until the leftmost bit position contains a one and correspondingly reducing the exponent by the number of positions shifted. For intermediate results in which coefficient overflow has occurred, the normalization process shall consist of right shifting the fraction one bit position and correspondingly increasing the exponent by one. For double precision floating point numbers, the entire fraction shall participate in the normalization such that the rightmost part may or may not appear as a normalized single precision number as determined by the value of the fraction.

For quotient and product instructions (Op. 32, 33, 36 and 37) if the operands are unnormalized, the results may be unnormalized. (See the individual instruction descriptions.)

When exponent arithmetic operations on standard floating numbers generate an intermediate exponent which is Out of Range, but normalization requirements generate an adjusted exponent which is no longer Out of Range, then neither Exponent Overflow nor Exponent Underflow shall be recorded for the final results.

2.4.1.5 Exceptions

With respect to floating point exceptions, (specifically Exponent Overflow, Exponent Underflow, Indefinite, and Loss of Significance), bit position assignments within the User Condition and User Mask Registers shall be in accordance with paragraphs 2.8.3 and 2.8.4 of this specification.

2.4.1.6 Double Precision Register Designators

The terms "Xk+1" and "Xj+1" shall be used to designate an X Register associated with the rightmost part of a double precision floating point number. When the leftmost part of a double precision floating point number, as designated by the terms "Xk" and "Xj" is associated with Register XF (in hexadecimal notation) the terms "Xk+1" and "Xj+1" shall be interpreted as designating Register X0. Notation designating the two registers holding the complete double precision floating point number is either XXk or XXj. (See 2.4.3.4 through 2.4.3.6.)

2.4.1.7 User Mask Bits

The form of the result to be stored for certain arithmetic operations involving any of the three exception conditions Exponent Overflow, Exponent Underflow and FP Loss of Significance is a function of the user mask bit associated with the exception. Tables 2.4-3 through 2.4-12 specify the results for each floating point operation (SUM, DIFF, PROD, QUOT) as a function of the input operands. These tables are arranged in pairs -- user mask clear, user mask set -- for each operation.

When both input operands are standard numbers or, one is a standard number and the other ±Z1 or ±Z2:

user mask clear selects a predetermined form for the result

+0 for result ±Z2	Exponent Underflow
+0 for result ±Z3	FP Loss of Significance
+∞ for result +INF	Exponent Overflow
−∞ for result −INF	Exponent Overflow

user mask set selects the generated result.

When both input operands are ±Z1 or ±Z2 or either operand is ±INF:

the user mask does not affect the result and the forms +0, +∞ or −∞ are stored.

(Note that when FP Indefinite is detected, the result stored is always of the form, +IND.)

2.4.1.8 Conversion (Int/FP)

The instructions within this subgroup shall provide the means for converting 64-bit words, contained in the X Registers, between integer and floating point formats.

2.4.1.8.1 Convert from Integer to Floating Point

Convert, Floating Point Xk formed from Integer Xj

3Ajk (Ref. 097)

This instruction shall convert the signed, two's complement, binary integer initially contained in the 64-bit positions of Register Xj to its equivalent, normalized floating point representation and shall transfer this 64-bit result to Register Xk. Integers outside of the range of -2^{48} through $2^{48}-1$ shall be truncated in the rightmost bit positions during conversion.

The integer initially contained in Register Xj shall be interpreted as having a magnitude (M) within the following range:

$-2^{63} \leq M \leq 2^{63}-1$

When the integer initially contained in Register Xj consists entirely of zeros, it shall be transferred without change to Register Xk.

2.4.1.8.2 Convert from Floating Point to Integer

Convert, Integer Xk formed from Floating Point Xj

3Bjk (Ref. 098)

This instruction shall convert the 64-bit floating point number initially contained in the Xj Register to a signed, two's complement, binary integer and shall transfer this 64-bit result to Register Xk. (The fractional part of the binary equivalent shall be lost as a result of truncation of the appropriate rightmost bits).

When the 64-bit floating point number initially contained in the Xk register:

1. Has an actual (unbiased) exponent which is less than or equal to zero, the result shall consist of 64 zeros. No exception conditions are recorded.
2. Has a coefficient consisting entirely of zeros, the result shall consist of 64 zeros. No exception conditions are recorded.
3. Is indefinite, a Floating Point Indefinite condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of this instruction shall be inhibited and program interruption shall occur (2.8.3.14). When the corresponding user mask bit is clear and/or traps are disabled, a result consisting of 64 zeros shall be stored and instruction execution completed.
4. Is infinite, an Arithmetic Loss of Significance condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of this instruction shall be inhibited and program interruption shall occur (2.8.3.15). When the corresponding user mask bit is clear and/or traps are disabled, a result consisting of 64 zeros shall be stored and instruction execution completed.

Floating point numbers with magnitude (M) shall be correctly converted provided such numbers are within the following range:

$-(2^{63}-2^{15}) \leq M \leq 2^{63}-2^{15}$

For integers outside of this range, the number transferred to Register Xk shall represent only the least significant, (rightmost) 64 bits of the actual result, and an Arithmetic Loss of Significance condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of this instruction shall be inhibited and program interruption shall occur. (Thus, such numbers shall be truncated to their leftmost positions). See subparagraph 2.8.3.15 of this specification.

2.4.1.9 Arithmetic

The instructions within this subgroup shall provide the means for performing arithmetic operations on floating point numbers to the extent described in the following subparagraphs.

2.4.1.9.1 Floating Point Sum/Difference

a. Floating Point Sum, Xk replaced by Xk plus Xj

30jk (Ref. 099)

b. Floating Point Difference, Xk replaced by Xk minus Xj

31 jk (Ref. 100)

Inputs: For the execution of these instructions, when either or both of the input arguments initially contained in Registers Xk and Xj consist of an Infinite (±INF) or Indefinite (±INDEF) floating point number, as defined in subparagraph 2.4.1.3 of this specification, the floating point result transferred to Register Xk shall consist of a nonstandard floating point number as defined by tables 2.4-3, 2.4-4, 2.4-5 and 2.4-6.

For the execution of these instructions, when both of the input arguments initially contained in Registers Xk and Xj consist of zero (±Z1, ±Z2) as described in 2.4.1.3, the floating point result transferred to Register Xk shall consist entirely of zeros (+0) and no Loss of Significance shall be recorded.

For those nonstandard input arguments for which an Infinite result is transferred to Register Xk, an Exponent Overflow condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of the instruction shall complete and program interruption shall occur. See subparagraph 2.8.3.11 of this specification.

For those nonstandard input arguments for which an Indefinite result is transferred to Register Xk, a Floating Point Indefinite condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of the instruction shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.14 of this specification.

In the absence of either input argument being Infinite (±INF) or Indefinite (±INDEF) or both input arguments being zero (±Z1, ±Z2), these instructions shall execute according to the following descriptions.

Exponent Equalization: The exponents of the two floating point numbers initially contained in the Xk and Xj Registers shall be algebraically compared and when they are equal, that common exponent shall be used as the intermediate exponent with neither of the associated coefficients shifted prior to coefficient arithmetic. However, when the exponents are not equal, the coefficient associated with the smaller exponent shall be shifted right, end-off, the number of bit positions designated by the difference between the exponents, up to a maximum of 48. Thus, the coefficients shall be aligned and the larger exponent shall be used as the intermediate exponent.

When the exponent difference is greater than 48, the larger exponent and its associated coefficient shall be used as the intermediate exponent and coefficient.

Coefficient Arithmetic: The two aligned coefficients, each consisting of a sign and a 48-bit fraction shall be added or subtracted, as determined by the operation code, with the coefficient associated with the Xj Register correspondingly treated as the addend or the subtrahend. The algebraic result shall consist of a signed coefficient having 48 bits of precision along with an overflow bit, and shall be referred to as the intermediate coefficient. (The overflow bit shall provide the required allowance for "true" addition, i.e., FP sum of coefficients having like signs and FP Difference between coefficients having unlike signs.)

Coefficient Overflow: When the overflow bit associated with the intermediate coefficient is a one, the 48 bits of precision associated with the intermediate coefficient shall be shifted one bit position right, end-off, with the overflow bit inserted into the vacated, leftmost bit position. The intermediate exponent shall be increased by one to adjust for this right shift of the coefficient and, provided the intermediate exponent does not overflow, the adjusted exponent along with its bias, and the normalized coefficient along with its sign, shall be transferred to the 64-bit position of Register Xk as the final result.

Exponent Overflow: When the adjustment of the intermediate exponent results in overflow, and exponent overflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Overflow mask bit contained in the User Mask Register. (See subparagraph 2.8.3.11 and paragraph 2.8.4 of this specification.)

When the corresponding mask bit is a one at the time the Exponent Overflow condition is recorded, the adjusted exponent along with its bias, and the normalized coefficient along with its sign (±INF), shall be transferred to the 64-bit positions of Register Xk as the final result. If the trap is enabled, then the execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.11 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Overflow condition is recorded, the nonstandard floating point number Infinite (±∞), as defined in paragraph 2.4.1.3 of this specification, shall be transferred to the 64-bit positions of Register Xk as the final result.

Loss of Significance: When the overflow bit and the 48 bits of precision associated with the intermediate coefficient consist entirely of zeros and one or both of the input operands consisted of a standard floating point number, then a Floating Point Loss of Significance condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Floating Point Loss of Significance mask bit contained in the User Mask Register. (See subparagraph 2.8.3.13 and paragraph 2.8.4 of this specification.)

When the corresponding mask bit is a one at the time the Floating Point Loss of Significance condition is recorded, the intermediate exponent along with its bias, and the intermediate coefficient along with its positive sign (+Z3), shall be transferred to the 64 bits of Register Xk as the final result. If the trap is enabled, then the execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.13 of this specification.

When the corresponding mask bit is a zero at the time the Floating Point Loss of Significance condition is recorded, the nonstandard floating point number zero (+0) as defined in subparagraph 2.4.1.3 of this specification, shall be transferred to the 64-bit positions of Register Xk as the final result.

Normalization: When the overflow bit associated with the intermediate coefficient is a zero and the 48 bits of precision associated with the intermediate coefficient do not consist entirely of zeros, these 48 bits of precision shall be left shifted to the extent required to achieve normalization, i.e., a one in the leftmost bit position. Left shifting shall be accomplished end-off, with zeros inserted on the right, for from 0 to 47 bit positions. For each bit position shifted left, the intermediate exponent shall be decreased by one. Upon completion of normalization, provided the exponent has not underflowed, the adjusted exponent along with its bias, and the normalized coefficient along with its sign shall be transferred to the 64-bit positions of Register Xk as the final result.

Exponent Underflow: When the adjustment of the exponent results in underflow, an Exponent Underflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Underflow mask bit contained in the User Mask Register. (See subparagraph 2.8.3.12 and paragraph 2.8.4 of this specification.)

When the corresponding mask bit is a one at the time the Exponent Underflow condition is recorded, the adjusted exponent along with its bias, and the normalized coefficient along with its sign (±Z2), shall be transferred to the 64-bit positions of the Xk Register as the final result. If the trap is enabled, then the execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.12 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Underflow condition is recorded, the nonstandard floating point number Zero (+0), as defined in paragraph 2.4.1.3 of this specification, shall be transferred to the 64-bit positions of Register Xk as the final result.

2.4.1.9.2 Floating Point Product

Floating Point Product, Xk replaced by Xk times Xj

32jk (Ref. 103)

Nonstandard Inputs: For the execution of this instruction, when either or both of the input arguments initially contained in Registers Xk and Xj consist of a Zero (±Z1, ±Z2), Infinite (±INF) or Indefinite (±INDEF) floating point number, as defined in subparagraph 2.4.1.3 of this specification, the floating point result transferred to Register Xk shall consist of a nonstandard floating point number as defined by tables 2.4-7 and 2.4-8.

For those nonstandard input arguments for which an Infinite result is transferred to Register Xk, an Exponent Overflow condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of the instruction shall complete and program interruption shall occur. See subparagraph 2.8.3.11 of this specification.

For those nonstandard input arguments for which an Indefinite result is transferred to Register Xk, a Floating Point Indefinite condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of the instruction shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.14 of this specification.

Standard Inputs: In the absence of nonstandard input arguments, this instruction shall execution according to the following descriptions.

Exponent Arithmetic: The signed exponents initially contained in Registers Xk and Xj shall be algebraically added and the result shall be used as the intermediate exponent.

Coefficient Arithmetic: The signed coefficient initially contained in Register Xk shall be multiplied by the signed coefficient initially contained in Register Xj. The result shall consist of an algebraically signed product having 96 bits of precision.

Normalization: When the leftmost bit of the 96 bits of precision associated with the product is a one, the sign and leftmost 48 bits of the product shall be used as the intermediate coefficient. When the leftmost bit of the 96 bits of precision associated with the product is a zero, that product shall be shifted left end-off one bit position, the sign and leftmost 48 bits of the shifted result shall be used as the intermediate coefficient and the intermediate exponent shall be decreased by one.

Exponent Overflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the overflow direction, an Exponent Overflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Overflow Mask bit contained in the User Mask register. See subparagraph 2.8.3.11 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Overflow condition is recorded, the adjusted exponent along with its bias, and the intermediate coefficient along with its sign, (±INF) shall be transferred to the 64-bit positions of Register X as the final result. If the trap is enabled, then the execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.11 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Overflow condition is recorded, the nonstandard floating point number Infinite (±∞), as defined in subparagraph 2.4.1.3 of this specification, shall be transferred to the 64-bit positions of Register Xk as the final result.

Exponent Underflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the underflow direction, an Exponent Underflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Underflow mask bit contained in the User Mask register. See subparagraph 2.8.3.12 and paragraph 2.8.4 of this specification.

2.4.1.9.3 Floating Point Quotient

Floating Point Quotient, Xk replaced by Xk divided by Xj

33jk (Ref. 104)

Nonstandard Inputs: For the execution of this instruction, when either or both of the input arguments initially contained in Registers Xk and Xj consist of a Zero (±Z1, ±Z2), Infinite (±INF) or Indefinite (±INDEF) floating point number, as defined in subparagraph 2.4.1.3 of this specification, the floating point result transferred to Register Xk shall consist of a nonstandard floating point number as defined by tables 2.4-9 through 2.4-12.

For those nonstandard arguments for which an Infinite result is transferred to Register Xk, an Exponent Overflow condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of the instruction shall complete and program interruption shall occur. See subparagraph 2.8.3.11 of this specification.

For those nonstandard input arguments for which an Indefinite result is transferred to Register Xk, a Floating Point Indefinite condition shall be detected. When the corresponding user mask bit is set and the trap enabled, execution of the instruction shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.14 of this specification.

When the Xj contains a nonstandard value of Zero (±Z1, ±Z2), the contents of Register Xk shall not be changed and Divide Fault condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.8 of this specification.

Standard Inputs: In the absence of nonstandard input arguments, these instructions shall execute according to the following descriptions.

Exponent Arithmetic: The signed exponent associated with the Xj Register shall be subtracted from the signed exponent associated with Xk Register and the signed result shall be referred to as the intermediate exponent.

Divide Fault: When the coefficient associated with the Xj Register is unnormalized and can be divided into the coefficient associated with the Xk Register by a factor equal to or greater than 2.0, the contents of Register Xk shall not be changed and a Divide Fault condition shall be detected. Further, when the coefficient of Xj consists entirely of zeros (±Z3), the contents of Register Xk shall not be changed and a Divide Fault condition shall be detected. When the corresponding user mask bit is set and the trap is enabled instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.8 of this specification.

In the event that a pair of operands is such that a Divide Fault is detected and such that the exponent arithmetic will produce Exponent Overflow or Underflow, the Divide Fault and only the Divide Fault shall be reported.

Coefficient Arithmetic: The signed coefficient associated with the Xj Register shall be divided into the signed coefficient associated with the Xk Register. The division shall be fractional, i.e., 48 zeros shall be appended rightmost to the signed coefficient associated with the Xk Register in order to obtain a dividend having 96 bits of precision. The results of the division shall consist of an algebraically signed quotient having 48 bits of precision and an overflow bit. (The overflow bit shall provide the required allowance for those cases in which the divisor can be divided into the dividend by a factor equal to or greater than 1.0 but less than 2.0.)

Normalization: When the overflow bit associated with the quotient is a zero, the sign and 48 bits of precision associated with the quotient shall be used as the intermediate coefficient. When the overflow bit associated with the quotient is a one, the 48 bits of precision associated with the quotient shall be shifted one bit position right, end-off, with the overflow bit inserted into the vacated leftmost bit position. The signed, 48-bit result shall be used as the intermediate coefficient and the intermediate exponent shall be increased by one to adjust for the right shift of the quotient.

Exponent Overflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the overflow direction, an Exponent Overflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Overflow Mask bit contained in the User Mask register. See subparagraph 2.8.3.11 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Overflow condition is recorded, the adjusted exponent along with its bias, and the intermediate coefficient along with its sign, (±INF) shall be transferred to the 64-bit positions of Register Xk as the final result. If the trap is enabled, then the execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.11 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Overflow condition is recorded, the nonstandard floating point number Infinite (±∞), as defined in subparagraph 2.4.1.3 of this specification, shall be transferred to the 64-bit positions of Register Xk as the final result.

Exponent Underflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the underflow direction, an Exponent Underflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Underflow mask bit contained in the User Mask register. See subparagraph 2.8.3.12 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Underflow condition is recorded, the adjusted exponent along with its bias and the intermediate coefficient along with its sign (±Z2) shall be transferred to the 64-bit positions of the Xk Register as the final result. If the trap is enabled, then the execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.12 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Underflow condition is recorded, the nonstandard floating point number Zero (+0) as defined in subparagraph 2.4.1.3 of this specification shall be transferred to the 64-bit positions of the Xk Register as the final result.

Result in Range: When the intermediate exponent, including the adjustment for normalization when applicable, is not equal to an Out of Range value, that intermediate exponent along with its bias, and the intermediate coefficient along with its sign, shall be transferred to the 64-bit positions of Register Xk as the final result. This final result shall always consist of a normalized number when both numbers initially contained in the Xk and Xj Registers consisted of normalized numbers.

2.4.1.9.4 Double Precision Floating Point Sum/Difference

a. Floating Point Sum, XXk replaced by XXk plus XXj

34 jk (Ref. 105)

b. Floating Point Difference, XXk replaced by XXk minus XXj

35 jk (Ref. 106)

Inputs: For the execution of these instructions, when either or both of the input arguments initially contained in Registers Xk and Xj consist of an Infinite (±INF) or Indefinite (±INDEF) floating point number, as defined in subparagraph 2.4.1.3 of this specification, the floating point result transferred to Registers Xk and Xk+1 shall consist of nonstandard floating point numbers as defined by tables 2.4-3, 2.4-4, 2.4-5 and 2.4-6.

For the execution of these instructions, when both of the input arguments initially contained in Registers Xk, Xk+1 and Xj, Xj+1 consist of zero (±Z1, ±Z2) as described in paragraph 2.4.1.3, the floating point result transferred to Registers Xk and Xk+1 shall consist entirely of zeros (+0) and no Loss of Significance shall be recorded.

For those input arguments for which an Infinite result is transferred to Registers Xk and Xk+1, an Exponent Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, execution of the instruction shall complete and program interruption shall occur. See subparagraph 2.8.3.11 of this specification.

For those input arguments for which an Indefinite result is transferred to Registers Xk and Xk+1 a Floating Point Indefinite condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, execution of the interaction shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.14 of this specification.

In the absence of either input argument being Infinite (±INF) or Indefinite (±INDEF) or both input arguments being Zero (±Z2, ±Z2) these instructions shall execute according to the following descriptions.

Exponent Equalization: The exponents of the two floating point numbers initially contained in the Xk and Xj Registers shall be algebraically compared and when they are equal, that common exponent shall be used as the intermediate exponent with neither of the associated coefficients shifted prior to coefficient arithmetic. However, when the exponents are not equal, the coefficient associated with the smaller exponent shall be shifted right, end-off, the number of bit positions designated by the difference between the exponents, up to a maximum of 96. Thus, the coefficients shall be aligned and the larger exponent shall be used as the intermediate exponent.

Coefficient Arithmetic: The two aligned coefficients, each consisting of a signed fraction having 96 bits of precision shall be added or subtracted, as determined by the operation code, with the coefficient associated with Registers Xj and Xj+1 correspondingly treated as the addend or the subtrahend. The algebraic result shall consist of a signed coefficient having 96 bits of precision along with an overflow bit, and shall be referred to as the intermediate coefficient. (The overflow bit shall provide the required allowance for "true" addition, i.e. FP Sum of coefficients having like signs and FP Difference between coefficients having unlike signs.)

Coefficient Overflow: When the overflow bit associated with the intermediate coefficient is a one, the 96 bits of precision associated with the intermediate coefficient shall be shifted one bit position right, end-off, with the overflow bit inserted into the vacated, leftmost bit position. The intermediate exponent shall be increased by one to adjust for this right shift of the coefficient. If the intermediate exponent does not overflow, the adjusted exponent along with its bias and the leftmost 48 bits of the normalized coefficient along with its sign shall be transferred to the 64-bit positions of Register Xk as the leftmost half of the final result; also, the adjusted exponent along with its bias and the rightmost 48 bits of the normalized coefficient along with its sign shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result.

Exponent Overflow: When the adjustment of the intermediate exponent results in overflow, an Exponent Overflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Overflow mask bit contained in the User Mask register. See subparagraph 2.8.3.11 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Overflow condition is recorded, the adjusted exponent along with its bias, and the leftmost 48 bits of the normalized coefficient along with its sign (±INF) shall be transferred to the 64-bit positions of Register Xk as the leftmost half of the final result; also, the adjusted exponent along with its bias, and the rightmost 48 bits of the normalized coefficient along with its sign (±INF) shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result. When the corresponding user mask bit is set and the trap is enabled, execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.2.11 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Overflow condition is recorded, the nonstandard floating point number Infinite ($\pm\infty$) as defined in subparagraph 2.4.1.3 of this specification shall be transferred to the 64-bit positions of both Register Xk and Register Xk+1 as the final result.

Loss of Significance: When the overflow bit and the 96 bits of precision associated with the intermediate coefficient consist entirely of zeros and one or both of the input operands consist of a standard floating point number, then a Floating Point Loss of Significance condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Floating Point Loss of Significance mask bit contained in the User Mask register. See subparagraph 2.8.3.13 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Floating Point Loss of Significance condition is recorded, the intermediate exponent along with its bias and the leftmost 48 bits of the intermediate coefficient along with its positive sign (+Z3) shall be transferred to the 64 bits of Register Xk as the leftmost half of the final result; also, the intermediate exponent along with its bias and the rightmost 48 bits of the intermediate coefficient along with its positive sign (+Z3) shall be transferred to the 64 bit positions of Register Xk+1 as the rightmost half of the final result. If the trap is enabled, then execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.13 of this specification.

When the corresponding mask bit is a zero at the time the Floating Point Loss of Significance condition is recorded, the nonstandard floating point number zero (+0) as defined in subparagraph 2.4.1.3 of this specification shall be transferred to the 64-bit positions of both Register Xk and Register Xk+1 as the final result.

Normalization: When the overflow bit associated with the intermediate coefficient is a zero and the 96 bits of precision associated with the intermediate coefficient do not consist entirely of zeros, these 96 bits of precision shall be left-shifted to the extent required to achieve normalization, i.e. a one in the leftmost bit position. Left shifting shall be accomplished end-off, with zeros inserted on the right, for from 0 to 95 bit positions. For each bit position shifted left, the intermediate exponent shall be decreased by one. Upon completion of normalization, provided the exponent has not underflowed, the adjusted exponent along with its bias and the leftmost 48 bits of the normalized coefficient along with its sign shall be transferred to the 64-bit positions of Register Xk as the leftmost half of the final result; also, the adjusted exponent along with its bias and the rightmost 48 bits of the normalized coefficient along with its sign shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result.

Exponent Underflow: When the adjustment of the exponent results in underflow, an Exponent Underflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Underflow mask bit contained in the User Mask register. See subparagraph 2.8.3.12 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Underflow condition is recorded, the adjusted exponent along with its bias and the leftmost 48 bits of the normalized coefficient along with its sign (±Z2) shall be transferred to the 64-bit positions of the Xk Register as the leftmost half of the final result; also, the adjusted exponent along with its bias and the rightmost 48 bits of the normalized coefficient along with its sign (±Z2) shall be transferred to the Register Xk+1 as the rightmost half of the final result. If the trap is enabled, then execution of this instruction shall complete and program interruption shall occur. See paragraph 2.8.3.12 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Underflow condition is recorded, the nonstandard floating point number Zero (+0) as defined in subparagraph 2.4.1.3 of this specification shall be transferred to the 64-bit positions of both Register Xk and Register Xk+1 as the final result.

2.4.1.9.5 Double Precision Floating Point Product

Floating Point Product, XXk replaced by XXk times XXj

36jk (Ref. 107)

Nonstandard Inputs: For the execution of this instruction, when either or both of the input arguments initially contained in Registers Xk and Xj consist of a Zero (±Z1, ±Z2), Infinite (±INF) or Indefinite (±INDEF) floating point number, as defined in subparagraph 2.4.1.3 of this specification, the floating point result transferred to Registers Xk and Xk+1 shall consist of nonstandard floating point numbers as defined by tables 2.4-7 and 2.4-8.

For those input arguments for which an Infinite result is transferred to Registers Xk and Xk+1, an Exponent Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, execution of the instruction shall complete and program interruption shall occur. See subparagraph 2.8.3.11 of this specification.

For those input arguments for which an Indefinite result is transferred to Registers Xk and Xk+1 a Floating Point Indefinite condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, execution of the instruction shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.14 of this specification.

Standard Inputs: In the absence of nonstandard input arguments, this instruction shall execute according to the following descriptions.

Exponent Arithmetic: The signed exponents initially contained in Register Xk and Xj shall be algebraically added and the result shall be used as the intermediate exponent.

Coefficient Arithmetic: The signed coefficient initially contained in Registers Xk and Xk+1 shall be multiplied by the signed coefficient initially contained in Registers Xj and Xj+1. The result shall consist of an algebraically signed product having 192 bits of precision.

Normalization: When the leftmost bit of the 192 bits of precision associated with the product is a one, the sign and leftmost 96 bits of the product shall be used as the intermediate coefficient. When the leftmost bit of the 192 bits of precision associated with the product is a zero, that product shall be shifted left end-off one bit position, the sign and leftmost 96 bits of the shifted result shall be used as the intermediate coefficient and the intermediate exponent shall be decreased by one.

Exponent Overflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the overflow direction, an Exponent Overflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Overflow Mask bit contained in the User Mask register. See subparagraph 2.8.3.11 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Overflow condition is recorded, the adjusted exponent along with its bias and the leftmost 48 bits of the intermediate coefficient along with its sign (±INF) shall be transferred to the 64-bit positions of Register Xk as the leftmost half of the final result; also, the adjusted exponent along with its bias and the rightmost 48-bit positions of the intermediate coefficient along with its sign (±INF) shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result. If the trap is enabled, then execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.11 of this specification.

When the corresponding mask bit is zero at the time the Exponent Overflow condition is recorded, the nonstandard floating point number Infinite (±∞) as defined in subparagraph 2.4.1.3 of this specification, shall be transferred to the 64-bit positions of both Register Xk and Register Xk+1 as the final result.

Exponent Underflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the underflow direction, an Exponent Underflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Underflow mask bit contained in the User Mask register. See subparagraph 2.8.3.12 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Underflow condition is recorded, the adjusted exponent along with its bias and the leftmost 48 bits of the intermediate coefficient along with its sign (±Z2) shall be transferred to the 64-bit positions of Register Xk as the leftmost half of the final result; also, the adjusted exponent along with its bias and the rightmost 48-bit positions of the intermediate coefficient along with its sign (±Z2) shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result. If the trap is enabled, then execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.12 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Underflow condition is recorded, the nonstandard floating point number Zero (+0), as defined in subparagraph 2.4.1.3 of this specification, shall be transferred to the 64-bit positions of both Register Xk and Register Xk+1 as the final result.

Result in Range: When the intermediate exponent, including the adjustment for normalization when applicable, is not equal to an Out of Range value, the intermediate exponent along with its bias and the leftmost 48 bits of intermediate coefficient along with its sign shall be transferred to the 64-bit positions of Register Xk as the leftmost half of the final result; also, the intermediate exponent along with its bias and the rightmost 48 bits of the intermediate coefficient along with its sign shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result.

2.4.1.9.6 Double Precision Floating Point Quotient

Floating Point Quotient, XXk replaced by XXk divided XXj

37jk (Ref. 108)

Nonstandard Inputs: For the execution of this instruction, when either or both of the input arguments initially contained in Registers Xk and Xj consist of a Zero (±Z1, ±Z2), Infinite (±INF) or Indefinite (±INDEF) floating point number, as defined in subparagraph 2.4.1.3 of this specification, the floating point result transferred to Registers Xk and Xk+1 shall consist of nonstandard floating point numbers as defined by tables 2.4-9 through 2.4-12.

For those input arguments for which an Infinite result is transferred to Registers Xk and Xk+1, an Exponent Overflow condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, execution of the instruction shall complete and program interruption shall occur. See subparagraph 2.8.3.11 of this specification.

For those input arguments for which an Indefinite result is transferred to Registers Xk and Xk+1, a Floating Point Indefinite condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, execution of the instruction shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.14 of this specification.

When the Xj Register contains a nonstandard value of Zero (±Z1, ±Z2), the contents of Registers Xk and Xk+1 shall not be changed and a Divide Fault condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.8 of this specification.

Standard Inputs: In the absence of nonstandard input arguments, this instruction shall execute according to the following descriptions:

Exponent Arithmetic: The signed exponent associated with the Xj Register shall be subtracted from the signed exponent associated with Xk Register and the signed result shall be referred to as the intermediate exponent.

Divide Fault: When the coefficient associated with the Xj Register is unnormalized and can be divided into the coefficient associated with the Xk Register by a factor equal to or greater than 2.0, the contents of Registers Xk and Xk+1 shall not be changed and a Divide Fault Condition shall be detected. Further, when the coefficients of Xj and Xj+1 consist entirely of zeros (±Z3), the contents of Register Xk and Xk+1 shall not be changed and a Divide Fault condition shall be detected. When the corresponding user mask bit is set and the trap is enabled, instruction execution shall be inhibited and program interruption shall occur. See subparagraph 2.8.3.8 of this specification.

In the event that a pair of operands is such that Divide Fault is detected and such that the exponent arithmetic will produce Exponent Overflow or Underflow, the Divide Fault and only the Divide Fault will be reported.

Coefficient Arithmetic: The signed coefficient associated with the Xj Register shall be divided into the signed coefficient associated with the Xk Register. The division shall be fractional, i.e., 96 zeros shall be appended rightmost to the signed coefficient associated with the Xk Register in order to obtain a dividend having 192 bits of precision. The results of the division shall consist of an algebraically signed quotient having 96 bits of precision and an overflow bit. (The overflow bit shall provide the required allowance for those cases in which the divisor can be divided into the dividend by a factor equal to or greater than 1.0 but less than 2.0.)

Normalization: When the overflow bit associated with the quotient is a zero, the sign and 96 bits of precision associated with the quotient shall be used as the intermediate coefficient. When the overflow bit associated with the quotient is a one, the 96 bits of precision associated with the quotient shall be shifted one bit position right, end-off, with the overflow bit inserted into the vacated leftmost bit position. The signed, 96-bit result shall be used as the intermediate coefficient and the intermediate exponent shall be increased by one to adjust for the right shift of the quotient.

Exponent Overflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the overflow direction, an Exponent Overflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Overflow Mask bit contained in the User Mask register. See subparagraph 2.8.3.11 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Overflow condition is recorded, the adjusted exponent along with its bias and the leftmost 48 bits of the intermediate coefficient along with its sign (±INF) shall be transferred to the 64 positions of Register Xk as the leftmost half of the final result; also, the adjusted exponent along with its bias and the rightmost 48-bit positions of the intermediate coefficient along with its sign (±INF) shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result. If the trap is enabled, then execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.11 of this specification.

When the corresponding mask bit is zero at the time the Exponent Overflow condition is recorded, the nonstandard floating point number Infinite (±∞) as defined in subparagraph 2.4.1.3 of this specification shall be transferred to the 64-bit positions of both Register Xk and Register Xk+1 as the final result.

Exponent Underflow: When the intermediate exponent, including the adjustment for normalization when applicable, is equal to an Out of Range value in the underflow direction, an Exponent Underflow condition shall be recorded and the final result of the associated instruction shall be determined according to the state of the Exponent Underflow mask bit contained in the User Mask Register. See subparagraph 2.8.3.12 and paragraph 2.8.4 of this specification.

When the corresponding mask bit is a one at the time the Exponent Underflow condition is recorded, the adjusted exponent along with its bias and the leftmost 48 bits of the intermediate coefficient along with its sign (±Z2) shall be transferred to the 64-bit positions of Register Xk as the leftmost half of the final result; also, the adjusted exponent along with its bias and the rightmost 48-bit positions of the intermediate coefficient along with its sign (±Z2) shall be transferred to the 64-bit positions of Register Xk+1 as the rightmost half of the final result. If the trap is enabled, then execution of the instruction shall complete and program interruption shall occur. See paragraph 2.8.3.12 of this specification.

When the corresponding mask bit is a zero at the time the Exponent Underflow condition is recorded, the nonstandard floating point number Zero (+0) as defined in subparagraph 2.4.1.3 of this specification shall be transferred to the 64-bit positions of both Register Xk and Register Xk+1 as the final result.

2.4.1.10 Branch

The instructions in this subgroup shall consist of conditional branch instructions.

Each of these conditional branch instructions shall perform a comparison between two floating point numbers. Then, based on the relationship between the results of that comparison and the branch condition as specified by means of the instruction's operation code, each conditional branch instruction shall perform either a normal exit or a branch exit.

Normal Exit: When the results of a comparison do not satisfy the branch condition as specified by the operation code, a normal exit shall be performed. A normal exit for all conditional branch instructions shall consist of adding four to the rightmost 32 bits of the PVA obtained from the P Register with that 32-bit sum returned to the P Register in its rightmost 32-bit positions.

Branch Exit: When the results of a comparison satisfy the branch condition as specified by the operation code, a branch exit shall be performed. A branch exit shall consist of expanding the 16-bit Q field from the instruction to 31 bits by means of sign extension, shifting these 31 bits left one bit position with a zero inserted on the right, and adding this 32-bit shifted result to the rightmost 32 bits of the PVA obtained from the P Register with the 32-bit sum returned to the P Register in its rightmost 32-bit positions.

2.4.1.10.1 Compare and Branch

a. Branch to P displaced by 2*Q if floating point Xj equal to Xk

98jkQ (Ref. 109)

b. Branch to P displaced by 2*Q if floating point Xj not equal to Xk

99jkQ (Ref. 110)

c. Branch to P displaced by 2*Q if floating point Xj greater than Xk

9AjkQ (Ref. 111)

d. Branch to P displaced by 2*Q if floating point Xj greater than or equal to Xk

9BjkQ (Ref. 112)

Operation: Each of these instructions shall perform an algebraic comparison of the 64-bit word obtained from Register Xj to the 64-bit word obtained from Register Xk. Each of these 64-bit words shall be treated as a signed single precision floating point number as described in subparagraph 2.4.1.1 of this specification. The contents of Register X0 shall be interpreted as consisting entirely of zeros with respect to both Xk and Xj.

Except for standard floating point numbers having like signs, the results of the comparisons for all of these instructions are given in table 2.4-2 of this specification. All comparisons for which the results are Indefinite, as indicated by "IND" in table 2.4-2, shall cause a Floating Point Indefinite condition to be recorded. When the corresponding user mask bit is clear and/or the trap is not enabled, the instruction shall perform a normal exit. When the trap is enabled and the corresponding mask bit is set, execution of the instruction shall be inhibited and program interruption shall occur. The PVA stored during the interrupt shall point to the Branch instruction that set the Floating Point Indefinite condition bit. See subparagraph 2.8.3.14 of this specification.

For standard floating point number having like signs, a floating point subtract shall be performed in the manner described in subparagraph 2.4.3.1 of this specification, with the exception that the operation is performed as if the (FP Overflow, Underflow and Loss of Significance) User Mask bits were set (Z2 not forced to zero, etc.) and that the result shall not be transferred to Register Xk but shall be interpreted in its post-normalized form to determine the results of the comparison.

These instructions shall perform a normal exit or a branch exit in the manner previously described in paragraph 2.2.3 of this specification.

2.4.1.10.2 Exception Branch

Branch to P displaced by 2*Q if floating point Xk is exception per j

9EjkQ (Ref. 113)

This instruction shall perform a branch exit in the manner previously described in paragraph 2.4.4 of this specification when the exception condition, as designated by the rightmost 2 bits of the j field from the instruction, is applicable to the 64-bit floating point number contained in the Xk Register.

This instruction shall perform a normal exit in the manner previously described in paragraph 2.4.4 of this specification when the exception condition, as designated by the rightmost 2 bits of the j field from the instruction, is not applicable to the 64-bit floating point number contained in the Xk Register.

The values of the rightmost 2 bits of the j field from the instruction shall be associated with exception conditions as follows:

if **00, Exponent Overflow**

nonstandard floating point numbers having biased exponents in the range: $5000 \leq exp \leq 6FFF$

01, Exponent Underflow

nonstandard floating point numbers having biased exponents in the range: $0000 \leq exp \leq 2FFF$

10 or 11, Indefinite

nonstandard floating point numbers having biased exponents in the range: $7000 \leq exp \leq 7FFF$

2.4.1.11 Compare

Compare floating point Xj to Xk, result to X1R

3Cjk (Ref. 114)

This instruction shall perform an algebraic comparison of the 64-bit word initially contained in Register Xj to the 64-bit word initially contained in Register Xk with the result transferred to Register X1 Right. Each of these 64-bit words shall be treated as a signed single precision floating point number as previously described in subparagraph 2.4.1.1 of this specification. The contents of Register X0 shall be interpreted as consisting entirely of zeros with respect to both the Xk and Xj Registers.

Except for standard floating point numbers having like signs, the results of the comparison are given in table 2.4-2 of this specification. All comparisons for which the results are indefinite shall cause a Floating Point Indefinite condition to be detected. When the corresponding user mask bit is clear and/or the trap is not enabled, register X1 Right shall be cleared in bit positions 33 through 63 and shall be set in bit position 32. When the trap is enabled and the corresponding mask bit is set, execution of the instruction shall be inhibited and program interruption shall occur. The PVA stored during the interrupt, however, shall point to the Compare instruction that set the Floating Point Indefinite condition bit. See subparagraph 2.8.3.14 of this specification.

For standard floating point numbers having like signs a floating point subtract shall be performed in the manner described in subparagraph 2.4.3.1 of this specification, with the exception that the operation is performed as if the (FP Overflow, Underflow and Loss of Significance) User Mask bits were set (Z2 not forced to zero, etc.) and that the result shall not be transferred to Register Xk but shall be interpreted in its post-normalized form to determine the result of the comparison.

When the initial contents of the Xj Register are equal to the initial contents of the Xk Register, Register X1 Right shall be cleared in all 32 bit positions.

When the initial contents of the Xj Register are greater than the initial contents of the Xk Register, Register X1 Right shall be cleared in bit positions 32 and 34 through 63 and shall be set in bit position 33.

When the initial contents of the Xj Register are less than the initial contents of the Xk Register, Register X1 Right shall be cleared in bit positions 34 through 63 and shall be set in bit positions 32 and 33.

2.4.1.12 Results

The symbols used in tables 2.4-2 through 2.4-12 are defined in table 2.4-1 and as follows:

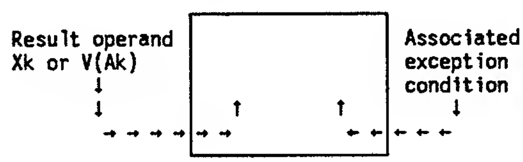
INDC - A result of indefinite returned by the floating point compare instruction. That is, a value for X1-Right = $(8000\ 0000)_{16}$.

These symbols represent standard numbers with nonzero coefficients; ±N but not ±Z3.

S - Algebraic sum of two floating point numbers.
D - Algebraic difference of two floating point numbers.
P - Algebraic product of two floating point numbers.
Q - Algebraic quotient of two floating point numbers.

DVF - The Divide Fault condition (UCR55).
OVL - The Exponent Overflow condition (UCR58).
UVL - The Exponent Underflow condition (UCR59).
LOS - The Floating Point Loss of Significance condition (UCR60).
IND - The Floating Point Indefinite condition (UCR61).

Key for Tables 2.4-3 through 2.4-12:



	Xk ↓ \ Xj	STANDARD NUMBERS				NONSTANDARD NUMBERS			
		+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+D,+Z2 < -D,-Z2 > +Z3 =	<	+D,+Z2 < +Z3 =	<	<	>	<	(See Note)
STANDARD	-N	>	+D,+Z2 < -D,-Z2 > +Z3 =	>	-D,-Z2 > +Z3 =	>	>	<	(See Note)
STANDARD	+Z3	-D,-Z2 > +Z3 =	<	+Z3 =	<	<	>	<	(See Note)
STANDARD	-Z3	>	+D,+Z2 < +Z3 =	>	+Z3 =	>	>	<	(See Note)
NONSTANDARD	±Z1 ±Z2	>	<	>	<	=	>	<	(See Note)
NONSTANDARD	+INF	<	<	<	<	<	(See Note)	<	(See Note)
NONSTANDARD	-INF	>	>	>	>	>	>	(See Note)	(See Note)
NONSTANDARD	±INDEF	(See Note)	(See Note)	(See Note)	(See Note)	(See Note)	(See Note)	(See Note)	(See Note)

Note: For the floating point compares indicated, FP Branch instructions (2.4.4.1) perform normal exit and record FP Indefinite (UCR61). FP Compare instructions (2.4.5) set X1 to $(8000\ 0000)_{16}$ and record FP Indefinite (UCR61), except when UCR61 is set and Traps are enabled in which case X1 is not altered.

Table 2.4-2. Floating Point Compare Results

Xk,V(Aj) ↓ / Xj,V(Ai) →		STANDARD NUMBERS				NONSTANDARD NUMBERS			
		+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+S +∞ OVL +0 UVL	±S +0 UVL +0 LOS	+S +0 UVL +0 LOS	+S +0 UVL +0 LOS	+N +0 UVL	+∞ OVL	-∞ OVL	+IND IND
	-N		-S -∞ OVL +0 UVL	-S +0 UVL +0 LOS	-S +0 UVL +0 LOS	-N +0 UVL	+∞ OVL	-∞ OVL	+IND IND
	+Z3			+0 LOS	+0 LOS	+0 LOS	+∞ OVL	-∞ OVL	+IND IND
	-Z3				+0 LOS	+0 LOS	+∞ OVL	-∞ OVL	+IND IND
NONSTANDARD	±Z1 ±Z2					+0	+∞ OVL	-∞ OVL	+IND IND
	+INF						+∞ OVL	+IND IND	+IND IND
	-INF							-∞ OVL	+IND IND
	±INDEF								+IND IND

Table 2.4-3. FP Sum Results $\left.\begin{cases} Xk \leftarrow Xk + Xj \\ V(Ak) \leftarrow V(Aj) + V(Ai) \end{cases}\right\}$ UM Clear

Xk,V(Aj) ↓ \ Xj,V(Ai) →		STANDARD NUMBERS +N	STANDARD NUMBERS -N	STANDARD NUMBERS +Z3	STANDARD NUMBERS -Z3	NONSTANDARD NUMBERS ±Z1 ±Z2	NONSTANDARD NUMBERS +INF	NONSTANDARD NUMBERS -INF	NONSTANDARD NUMBERS ±INDEF
STANDARD	+N	+S +INF OVL +Z2 UVL	±S ±Z2 UVL +Z3 LOS	+S +Z2 UVL +Z3 LOS	+S +Z2 UVL +Z3 LOS	+N +Z2 UVL	+∞ OVL	-∞ OVL	+IND IND
STANDARD	-N		-S -INF OVL -Z2 UVL	-S -Z2 UVL +Z3 LOS	-S -Z2 UVL +Z3 LOS	-N -Z2 UVL	+∞ OVL	-∞ OVL	+IND IND
STANDARD	+Z3			+Z3 LOS	+Z3 LOS	+Z3 LOS	+∞ OVL	-∞ OVL	+IND IND
STANDARD	-Z3				+Z3 LOS	+Z3 LOS	+∞ OVL	-∞ OVL	+IND IND
NONSTANDARD	±Z1 ±Z2					+0	+∞ OVL	-∞ OVL	+IND IND
NONSTANDARD	+INF						+∞ OVL	+IND IND	+IND IND
NONSTANDARD	-INF							-∞ OVL	+IND IND
NONSTANDARD	±INDEF								+IND IND

Table 2.4-4. FP Sum Results $\left.\begin{cases} Xk \leftarrow Xk + Xj \\ V(Ak) \leftarrow V(Aj) + V(Ai) \end{cases}\right\}$ UM Set

Traps Enabled:
- Scalar - Replace +IND with Xk
- Vector - Chart is as shown

Traps Disabled:
- Scalar and Vector - Chart is as shown

Xj,V(Ai) → Xk,V(Aj) ↓		STANDARD NUMBERS +N	STANDARD NUMBERS -N	STANDARD NUMBERS +Z3	STANDARD NUMBERS -Z3	NONSTANDARD NUMBERS ±Z1 ±Z2	NONSTANDARD NUMBERS +INF	NONSTANDARD NUMBERS -INF	NONSTANDARD NUMBERS ±INDEF
STANDARD	+N	±D +0 UVL +0 LOS	+D +∞ OVL +0 UVL	+D +0 UVL +0 LOS	+D +0 UVL +0 LOS	+N +0 UVL	-∞ OVL	+∞ OVL	+IND IND
STANDARD	-N	-D -∞ OVL +0 UVL	±D +0 UVL +0 LOS	-D +0 UVL +0 LOS	-D +0 UVL +0 LOS	-N +0 UVL	-∞ OVL	+∞ OVL	+IND IND
STANDARD	+Z3	-D +0 UVL +0 LOS	+D +0 UVL +0 LOS	+0 LOS	+0 LOS	+0 LOS	-∞ OVL	+∞ OVL	+IND IND
STANDARD	-Z3	-D +0 UVL +0 LOS	+D +0 UVL +0 LOS	+0 LOS	+0 LOS	+0 LOS	-∞ OVL	+∞ OVL	+IND IND
NONSTANDARD	±Z1 ±Z2	-N +0 UVL	+N +0 UVL	+0 LOS	+0 LOS	+0	-∞ OVL	+∞ OVL	+IND IND
NONSTANDARD	+INF	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+IND IND	+∞ OVL	+IND IND
NONSTANDARD	-INF	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	+IND IND	+IND IND
NONSTANDARD	±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND

Table 2.4-5. FP Difference Results $\left\{\begin{array}{l} Xk \leftarrow Xk - Xj \\ V(Ak) \leftarrow V(Aj) - V(Ai) \end{array}\right\}$ UM Clear

Xj,V(Ai) → Xk,V(Aj) ↓		STANDARD NUMBERS				NONSTANDARD NUMBERS			
		+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+D ±Z2 UVL +Z3 LOS	+D +INF OVL +Z2 UVL	+D +Z2 UVL +Z3 LOS	+D +Z2 UVL +Z3 LOS	+N +Z2 UVL	-∞ OVL	+∞ OVL	+IND IND
	-N	-D -INF OVL -Z2 UVL	+D ±Z2 UVL +Z3 LOS	-D -Z2 UVL +Z3 LOS	-D -Z2 UVL +Z3 LOS	-N -Z2 UVL	-∞ OVL	+∞ OVL	+IND IND
	+Z3	-D -Z2 UVL +Z3 LOS	+D +Z2 UVL +Z3 LOS	+Z3 LOS	+Z3 LOS	+Z3 LOS	-∞ OVL	+∞ OVL	+IND IND
	-Z3	-D -Z2 UVL +Z3 LOS	+D +Z2 UVL +Z3 LOS	+Z3 LOS	+Z3 LOS	+Z3 LOS	-∞ OVL	+∞ OVL	+IND IND
NONSTANDARD	±Z1 ±Z2	-N -Z2 UVL	+N +Z2 UVL	+Z3 LOS	+Z3 LOS	+0	-∞ OVL	+∞ OVL	+IND IND
	+INF	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+∞ OVL	+IND IND	+∞ OVL	+IND IND
	-INF	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	-∞ OVL	+IND IND	+IND IND
	±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND	+IND IND

Table 2.4-6. FP Difference Results $\left\{\begin{array}{l} Xk \leftarrow Xk - Xj \\ V(Ak) \leftarrow V(Aj) - V(Ai) \end{array}\right\}$ UM Set

Traps Enabled:
- Scalar - Replace +IND with Xk
- Vector - Chart is as shown

Traps Disabled:
- Scalar and Vector - Chart is as shown

Xj,V(Ai) / Xk,V(Aj)		STANDARD NUMBERS				NONSTANDARD NUMBERS			
		+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+P +∞ OVL +0 UVL +Z3	-P -∞ OVL +0 UVL -Z3	+∞ OVL +0 UVL +Z3	-∞ OVL +0 UVL -Z3	+0	+∞ OVL	-∞ OVL	+IND IND
	-N		+P +∞ OVL +0 UVL +Z3	-∞ OVL +0 UVL -Z3	+∞ OVL +0 UVL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
	+Z3			+∞ OVL +0 UVL +Z3	-∞ OVL +0 UVL -Z3	+0	+∞ OVL	-∞ OVL	+IND IND
	-Z3				+∞ OVL +0 UVL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
NONSTANDARD	±Z1 ±Z2					+0	+IND IND	+IND IND	+IND IND
	+INF						+∞ OVL	-∞ OVL	+IND IND
	-INF							-∞ OVL	+IND IND
	±INDEF								+IND IND

Table 2.4-7. FP Product Results $\left\{\begin{array}{l} Xk \leftarrow Xk \times Xj \\ V(Ak) \leftarrow V(Aj) \times V(Ai) \end{array}\right\}$ UM Clear

Xk,V(Aj) ↓ \ Xj,V(Ai) →		STANDARD NUMBERS +N	-N	+Z3	-Z3	NONSTANDARD NUMBERS ±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+P +INF OVL +Z2 UVL +Z3	-P -INF OVL -Z2 UVL -Z3	+INF OVL +Z2 UVL +Z3	-INF OVL -Z2 UVL -Z3	+0	+∞ OVL	-∞ OVL	+IND IND
	-N		+P +INF OVL +Z2 UVL +Z3	-INF OVL -Z2 UVL -Z3	+INF OVL +Z2 UVL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
	+Z3			+INF OVL +Z2 UVL +Z3	-INF OVL -Z2 UVL -Z3	+0	+∞ OVL	-∞ OVL	+IND IND
	-Z3				+INF OVL +Z2 UVL +Z3	+0	-∞ OVL	+∞ OVL	+IND IND
NONSTANDARD	±Z1 ±Z2					+0	+IND IND	+IND IND	+IND IND
	+INF						+∞ OVL	-∞ OVL	+IND IND
	-INF							+∞ OVL	+IND IND
	±INDEF								+IND IND

Table 2.4-8. FP Product Results $\left\{\begin{array}{l} Xk \leftarrow Xk \times Xj \\ V(Ak) \leftarrow V(Aj) \times V(Ai) \end{array}\right\}$ UM Set

Traps Enabled:
- Scalar - Replace +IND with Xk
- Vector - Chart is as shown

Traps Disabled:
- Scalar and Vector - Chart is as shown

	Xk ↓ \ Xj	STANDARD NUMBERS +N	-N	+Z3	-Z3	NONSTANDARD NUMBERS ±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+Q +∞ OVL +0 UVL Xk DVF	-Q -∞ OVL +0 UVL Xk DVF	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
	-N	-Q -∞ OVL +0 UVL Xk DVF	+Q +∞ OVL +0 UVL Xk DVF	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
	+Z3	+∞ OVL +0 UVL +Z3	-∞ OVL +0 UVL -Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
	-Z3	-∞ OVL +0 UVL -Z3	+∞ OVL +0 UVL +Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
NONSTANDARD	±Z1 ±Z2	+0	+0	+0	+0	Xk DVF	+0	+0	+IND IND
	+INF	+∞ OVL	-∞ OVL	+∞ OVL	-∞ OVL	Xk DVF	+IND IND	+IND IND	+IND IND
	-INF	-∞ OVL	+∞ OVL	-∞ OVL	+∞ OVL	Xk DVF	+IND IND	+IND IND	+IND IND
	±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	Xk DVF	+IND IND	+IND IND	+IND IND

Table 2.4-9. FP Quotient Results Xk ← Xk / Xj UM Clear

	Xj → Xk ↓	STANDARD NUMBERS				NONSTANDARD NUMBERS			
		+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+Q +INF OVL +Z2 UVL Xk DVF	-Q -INF OVL -Z2 UVL Xk DVF	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
STANDARD	-N	-Q -INF OVL -Z2 UVL Xk DVF	+Q +INF OVL +Z2 UVL Xk DVF	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
STANDARD	+Z3	+INF OVL +Z2 UVL +Z3	-INF OVL -Z2 UVL -Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
STANDARD	-Z3	-INF OVL -Z2 UVL -Z3	+INF OVL +Z2 UVL +Z3	Xk DVF	Xk DVF	Xk DVF	+0	+0	+IND IND
NONSTANDARD	±Z1 ±Z2	+0	+0	+0	+0	Xk DVF	+0	+0	+IND IND
NONSTANDARD	+INF	+∞ OVL	-∞ OVL	+∞ OVL	-∞ OVL	Xk DVF	+IND IND	+IND IND	+IND IND
NONSTANDARD	-INF	-∞ OVL	+∞ OVL	-∞ OVL	+∞ OVL	Xk DVF	+IND IND	+IND IND	+IND IND
NONSTANDARD	±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	Xk DVF	+IND IND	+IND IND	+IND IND

Table 2.4-10. FP Quotient Results Xk ← Xk / Xj UM Set

Traps Enabled:
Replace +IND with Xk

Traps Disabled:
Chart is as shown

V(Aj) ↓ \ V(Ai) →		STANDARD NUMBERS				NONSTANDARD NUMBERS			
		+N	-N	+Z3	-Z3	±Z1 ±Z2	+INF	-INF	±INDEF
STANDARD	+N	+Q +∞ OVL +0 UVL +IND DVF & IND	-Q -∞ OVL +0 UVL +IND DVF & IND	+IND DVF & IND	+IND DVF & IND	+IND DVF & IND	+0	+0	+IND IND
STANDARD	-N	-Q -∞ OVL +0 UVL +IND DVF & IND	+Q -∞ OVL +0 UVL +IND DVF & IND	+IND DVF & IND	+IND DVF & IND	+IND DVF & IND	+0	+0	+IND IND
STANDARD	+Z3	+∞ OVL +0 UVL +Z3	-∞ OVL +0 UVL -Z3	+IND DVF & IND	+IND DVF & IND	+IND DVF & IND	+0	+0	+IND IND
STANDARD	-Z3	-∞ OVL +0 UVL -Z3	+∞ OVL +0 UVL +Z3	+IND DVF & IND	+IND DVF & IND	+IND DVF & IND	+0	+0	+IND IND
NONSTANDARD	±Z1 ±Z2	+0	+0	+0	+0	+IND DVF & IND	+0	+0	+IND IND
NONSTANDARD	+INF	+∞ OVL	-∞ OVL	+∞ OVL	-∞ OVL	+IND DVF & IND	+IND IND	+IND IND	+IND IND
NONSTANDARD	-INF	-∞ OVL	+∞ OVL	-∞ OVL	+∞ OVL	+IND DVF & IND	+IND IND	+IND IND	+IND IND
NONSTANDARD	±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	+IND DVF & IND	+IND IND	+IND IND	+IND IND

Note: See 2.12.1.5

Table 2.4-11. FP Quotient Results V(Ak) ← V(Aj) / V(Ai) UM Clear

	V(Ai) → V(Aj) ↓	STANDARD NUMBERS +N	STANDARD NUMBERS -N	STANDARD NUMBERS +Z3	STANDARD NUMBERS -Z3	NONSTANDARD NUMBERS ±Z1 ±Z2	NONSTANDARD NUMBERS +INF	NONSTANDARD NUMBERS -INF	NONSTANDARD NUMBERS ±INDEF
STANDARD	+N	+Q +INF OVL +Z2 UVL +IND {DVF & IND}	-Q -INF OVL -Z2 UVL +IND {DVF & IND}	+IND {DVF & IND}	+IND {DVF & IND}	+IND {DVF & IND}	+0	+0	+IND IND
STANDARD	-N	-Q -INF OVL -Z2 UVL +IND {DVF & IND}	+Q +INF OVL +Z2 UVL +IND {DVF & IND}	+IND {DVF & IND}	+IND {DVF & IND}	+IND {DVF & IND}	+0	+0	+IND IND
STANDARD	+Z3	+INF OVL +Z2 UVL +Z3	-INF OVL -Z2 UVL -Z3	+IND {DVF & IND}	+IND {DVF & IND}	+IND {DVF & IND}	+0	+0	+IND IND
STANDARD	-Z3	-INF OVL -Z2 UVL -Z3	+INF OVL +Z2 UVL +Z3	+IND {DVF & IND}	+IND {DVF & IND}	+IND {DVF & IND}	+0	+0	+IND IND
NONSTANDARD	±Z1 ±Z2	+0	+0	+0	+0	+IND {DVF & IND}	+0	+0	+IND IND
NONSTANDARD	+INF	+∞ OVL	-∞ OVL	+∞ OVL	-∞ OVL	+IND {DVF & IND}	+IND IND	+IND IND	+IND IND
NONSTANDARD	-INF	-∞ OVL	+∞ OVL	-∞ OVL	+∞ OVL	+IND {DVF & IND}	+IND IND	+IND IND	+IND IND
NONSTANDARD	±INDEF	+IND IND	+IND IND	+IND IND	+IND IND	+IND {DVF & IND}	+IND IND	+IND IND	+IND IND

Note: See 2.12.1.5

Table 2.4-12. FP Quotient Results $V(Ak) \leftarrow V(Aj) / V(Ai)$ UM Set

Traps Enabled or Disabled:
Chart is as shown

2.4.2 Format: 32-Bit

The 32-bit floating point format, called half-precision, is not implemented in the CYBER 180 architecture.

2.5 LOGICAL ENVIRONMENT

A logical environment shall be defined by two sets of registers. The first set shall be referred to as the Processor State Register and shall include all items which are not unique to a process. Each processor shall have one set of Processor State Registers.

The second set of registers shall be referred to as the Process State Registers and shall include all items which are unique to a process. The act of going from one process state to another shall be referred to as an exchange. The contents of the Process State Registers associated with the exchange shall be referred to as an Exchange Package. Therefore, each process shall have one Exchange Package to define its unique environment.

2.5.1 Processor State Registers

See Table 2.5-1 and the following paragraphs for the definition of each of the Processor State Registers.

Processor State Register	Bit Positions (inclusive)
Job Process State	32 - 63
Monitor Process State	32 - 63
Page Table Address	32 - 63
Page Table Length	56 - 63
Page Size Mask	57 - 63
Element Identifier	32 - 63
System Interval Timer	32 - 63
Processor Identification	56 - 63
Virtual Machine Capability List	48 - 63
Keypoint Buffer Pointer	00 - 63

Table 2.5-1. Bit positions of Processor State Registers when copied to or from a 64-bit X Register

2.5.1.1 Job Process State (JPS)

The JPS shall consist of a 32-bit real memory byte address. It shall point to the first entry in the exchange package for the job process. The JPS address shall be aligned with bits 32 through 63 of real memory addresses. The JPS address shall be interpreted as zero, modulo 16. The processor shall ignore bit 32 and shall interpret bits 60, 61, 62 and 63 as zero. The writing of any value other than zero into these bits shall cause undefined operation with respect to the subsequent value of these bits. The processor shall always return zero on a read of these bits assuming the previous write was zero as described in 2.1.3.5b. (Also see table 2.5-2.)

A JPS which has bit 33 set causes undefined processor operation when this address is used.

Note: See 3.1.3 for the definition of a real memory address.

2.5.1.2 Monitor Process State (MPS)

The MPS shall consist of a 32-bit real memory byte address. It shall point to the first entry in the exchange package for the monitor process. The MPS address shall be aligned with bits 32 through 63 of real memory addresses. The MPS address shall be interpreted as zero, modulo 16. The processor shall ignore bit 32 and shall interpret bits 60, 61, 62, and 63 as zero. The writing of any value other than zero into these bits shall cause undefined operation with respect to the subsequent value of these bits. The processor shall always return zero on a read of these bits assuming the previous write was zero as described in 2.1.3.5b. (Also see table 2.5-2.)

A MPS which has bit 33 set causes undefined processor operation when this address is used.

2.5.1.3 Page Table Address (PTA)

The PTA shall consist of a 32-bit real memory byte address. It shall point to the first entry in the Page Table. The PTA address shall be aligned with bits 32 through 63 of real memory addresses. The processor may assume, when performing virtual address translations, that the Page Table Address is 0, modulo the Page Table Length. A PTA which is nonzero, modulo the PTL will cause undefined processor operation when virtual address translation is attempted. Note that the processor shall be halted when writing these registers, thus, no virtual address translations will be attempted until the processor is restarted. (See tables 2.1-2, 2.5-2.) The writing of any value other than zero into bits 52 through 63 shall cause undefined operation with respect to the subsequent value of these bits. The processor shall return the following on read operations:

Bit 32 - zero, assuming zero on previous write
Bits 33 through 51 - the contents of PTA as written
Bits 52 through 63 - zero assuming zero on previous write

A PTA which has bit 33 set causes undefined process operation when this address is used.

2.5.1.4 Page Table Length (PTL)

The PTL shall consist of a mask which shall specify the length of the Page Table. The PTL mask shall express the page table length 0, modulo 4096 bytes. The mask shall consist of a contiguous string of one bits, beginning in the rightmost bit position of the PTL Register and extending towards the leftmost bit position of the PTL Register.

There are two versions of the PTL register: 8 and 14 bits. The extended 14-bit PTL allows the smaller page sizes to be used on central memories up to 2 gigabytes. (See 1.6 for a list of specific models, and 3.5.2.1 for a further description of PTL). Table 3.5-2 indicates the expected limits for operation with the 8-bit PTL. The 4K pages can be used for 128 megabyte or smaller central memories, the 8K pages can be used for 256 megabyte or smaller central memories, and so forth.

```
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63

 X  0|<----------------------------MPS-------------------------------------------->| Z  Z  Z  Z
 X  0|<----------------------------JPS-------------------------------------------->| Z  Z  Z  Z

 X  0|<-----------------PTA------------------------------->| 0  0  0  0  0  0  0  0  0  0  0  0  PTL=512
 X  0|<------------------------------------------------>| 0  0  0  0  0  0  0  0  0  0  0  0  0       1K
 X  0|<--------------------------------------------->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0       2K
 X  0|<------------------------------------------>| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0       4K
 X  0|<--------------------------------------->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0       8K   8-bit
 X  0|<------------------------------------>| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0      16K    PTL
 X  0|<--------------------------------->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0      32K
 X  0|<------------------------------>| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0      64K
 X  0|<-----------PTA------------->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0     128K

32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63         14-bit
                                                                                                          PTL
 X  0|<-----------PTA---------->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 PTL=256K
 X  0|<--------------------->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0     512K
 X  0|<------------------>| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0       1M
 X  0|<--------------->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0       2M
 X  0|<------------>| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0       4M
 X  0|<---PTA--->| 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0       8M

 X  0|<----------------------------STA----------------------------------------------->| Z  Z  Z

                     X = Bit is ignored and not used by the processor
                     Z = Bit is ignored and zero is used by the processor
                     0 = Bit must be zero or processor operation is undefined
```

Table 2.5-2. Real Memory Addresses

2.5.1.5 Page Size Mask (PSM)

The PSM shall consist of a 7-bit mask which shall specify the page size used in allocating real central memory. (See subparagraph 3.4.2.2.)

2.5.1.6 Element Identifier (EID)

The EID shall consist of 32 bits and shall uniquely identify each hardware element, worldwide. See paragraph 1.5 for the format of the EID register.

2.5.1.7 System Interval Timer (SIT)

The SIT shall be a 32-bit counter which the system may use to establish a maximum time interval for job mode execution. See subparagraphs 2.5.3.2 and 2.8.1.12 of this specification.

2.5.1.8 Processor Identifier (PID)

The PID shall consist of 8 bits and shall uniquely identify the processors in a system as follows:

Processor	PID (Hex)
Primary processor	00
Optional processor	01

2.5.1.9 Virtual Machine Capability List (VMCL)

The VMCL shall consist of 16 bits which reflect the processor's virtual machine capabilities on a bit-by-bit basis as follows:

Bit 48 : CYBER 180
Bit 49 : *CYBER 170 Mode*
Bit 50 : Reserved
↓ ↓ ↓
↓ ↓ ↓
Bit 63 : Reserved

The processor shall provide zeros for bits 50 through 63 when VMCL is read via the maintenance channel.

2.5.1.10 Keypoint Buffer Pointer (KBP)

This 64-bit register shall contain the address (PVA) of the next location in central memory for writing keypoint data. Bits 0-15 are unused (see 2.1.3.5). Bits 16-63 contain the PVA. The loading of this register by means of the Copy to State Register instruction (0F) shall copy the rightmost bits of register Xk directly. There is no special test performed on the ring number portion of the PVA.

2.5.2 Process State Registers

Each Process State shall be defined by an individual Exchange Package. An Exchange Package shall consist of 52 64-bit words in central memory at contiguous word locations. The contents of an Exchange Package shall be formatted according to this specification such that corresponding interpretation by a processor shall provide the means for establishing a unique Process State.

Each Exchange Package in central memory shall contain Process State information in sufficient quantity and detail such that a processor may be dynamically switched between Exchange Packages. Moreover, when a processor is switched from a first Exchange Package to a second Exchange Package and at some later time is switched back to the first Exchange Package, the integrity of the processing which occurs for the Process State represented by the first Exchange Package shall not be affected.

Processors may on a model-dependent basis load any or all of the Exchange Package from central memory when a process is activated. To allow this freedom in implementation, the following items must be noted concerning the Exchange Package area associated with an active process:

- The contents of the Exchange Package in central memory are undefined.
- The contents of the Exchange Package in central memory must not be altered by another processor or I/O operation, or undefined processor execution will occur.
- The address register for the Exchange Package must not be altered while the associated process is active.

Those items in the Exchange Package which shall exist in registers when an Exchange Package is active shall be processor model-dependent. The processor model-dependent specifications shall define those items.

Figure 2.5-1 defines the contents of the first 52 words in an Exchange Package. The sections which follow shall define the items contained in those words.

a. The unused bits in the exchange package (see following list) are undefined when an exchange package is created and are ignored when an exchange package is loaded into the processor.

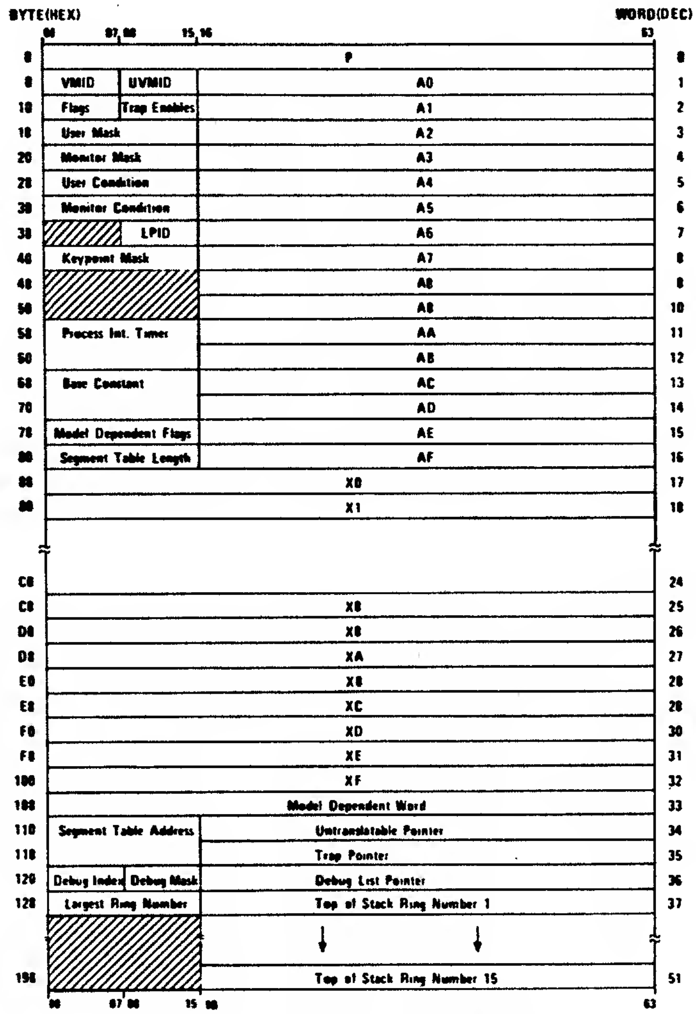
Word 1:	Bits 00, 01, 08 and 09	(Unused)
Word 2:	Bits 05 through 13	(Unused)
Word 7:	Bits 00 through 07	(Unused)
Word 9:	Bits 00 through 15	(Unused)
Word 10:	Bits 00 through 15	(Unused)
Word 16:	Bits 00 through 03	(Unused)
Word 36:	Bits 06 through 08	(Unused)
Word 37:	Bits 00 through 11	(Unused)
Words 38-51:	Bits 00 through 15	(Unused)

b. The statements made in item a. shall also apply to the Exchange Package, Word 3, Bits 00 through 06 (leftmost 7-bit positions of the User Mask) with the exception that these bits shall be treated as ones.

c. The modification of Process State Register values in a central memory exchange package by one processor at the time that process is being executed by another processor, shall result in undefined operation. Overlapped exchange packages in central memory may also result in undefined operations.

d. The C180 Exchange operations use RMAs to address the exchange packages (at JPS and at MPS) while cache, when present, is always addressed by SVA. The C180 exchange packages may either be in:

- cache bypass segments thus preventing stale data from appearing in the cache, or in
- noncache bypass segments thus requiring the software to purge cache appropriately to prevent stale data.



DETAIL FOR C180 EXCHANGE PACKAGE

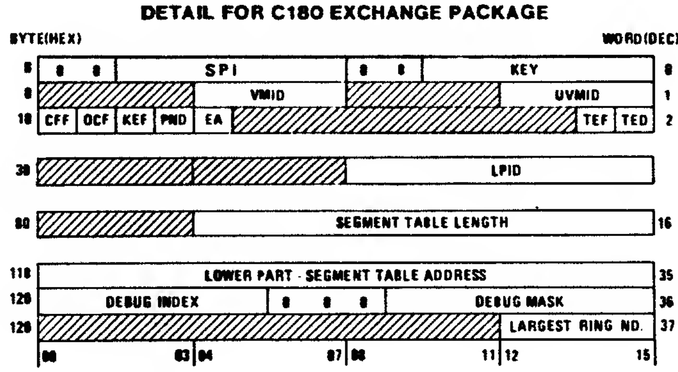


Figure 2.5-1. CYBER 180 Exchange Package (C180 Process)

2.5.2.1 Program Address Register (P)

See paragraph 2.1.1.1 for the definition of the P Register's contents.

P shall be located in bits 00 through 63 of word 0 in the Exchange Package.

2.5.2.2 A Registers

The 16 A Registers, A0 through AF, shall be located in bits 16 through 63 of words 1 through 16, respectively, in the Exchange Package. See paragraph 2.1.1.2 for the definition of the A Register's contents.

2.5.2.3 X Registers

The 16 X Registers, X0 through XF, shall be located in bits 00 through 63 of words 17 through 32, respectively, in the Exchange Package. See paragraph 2.1.1.3 for the definition of the X Register's contents.

2.5.2.4 Not Assigned

This paragraph is left blank intentionally.

2.5.2.5 Flags

The Flags field shall consist of five separate single bit flags which have the following definitions.

a. Critical Frame Flag (CFF)

   The CFF, if set, shall indicate that the currently active stack frame for the process defined by this Exchange Package is a "critical frame." In this context, software shall have exclusive control over the state of CFF.

   CFF shall be located in bit 0 of word 2 in the Exchange Package. (See 2.6.5.2 and 2.8.10.)

b. On Condition Flag (OCF)

   The OCF is intended to facilitate the handling of "on condition" traps on the part of the "process monitor." In this context, software shall have exclusive control over the state of OCF.

   OCF shall be located in bit 1 of word 2 in the Exchange Package. (See 2.6.5.2 and 2.8.10.)

c. Keypoint Enable Flag (KEF)

   The KEF, if set, shall enable the recording of keypoint data into the central memory location specified by the Keypoint Buffer Pointer. (See 2.5.1.15 and 2.6.1.7.)

   KEF shall be located in bit 2 of word 2 in the Exchange Package. (See 2.8.10.)

d. Process Not Damaged (PND)

Process Not Damaged is the primary determination made during CPU uncorrected error analysis. If Stop on Error is implemented, the most expeditious stop is executed in order to preserve the accuracy of the error and environmental data. There may not be time for the hardware to monitor all conditions that could result in a damaged process. Therefore, additional model-dependent status may be required to make a software Process Not Damaged decision.

For systems in which stop on error is not implemented, the PND flag is the result of a complete hardware analysis.

In either case, the definition of Process Not Damaged means the same thing, i.e. the ability of the process to be retried. If PND = true, software will retry the process to determine if the error is transient/intermittent or solid. If PND = false, software will abort the process.

The PND, when set during a C180 Job to Monitor exchange operation caused by an uncorrectable error, indicates that there is no evidence that the process being executed was damaged, and that it may be restarted. The PVA in P of the Exchange Package is the proper address to restart the process but is not necessarily the address of the instruction which initiated the activity that resulted in the malfunction. This flag is intended to allow recovery of job mode processes where possible.

Note that the default state of this flag is interpreted as process damaged. Thus, on a model-dependent basis, the hardware may detect many, few, or none of the undamaged processes and set PND accordingly. While a processor may report undamaged processes as damaged, it shall be a design objective to never report damaged processes as undamaged. Thus a process denoted as undamaged via PND shall be as likely to be correct as any process with no fault indication. This flag shall be ignored by the hardware when loading a C180 Exchange Package and is only defined in the Exchange Package resulting from a Detected Uncorrectable Error Interrupt.

PND shall be located in bit 3 of word 2 in the Exchange Package.

e. ECS Authorized (EA)

The EA, if set, shall enable the currently active process when in C170 State to access the ECS via the 011, 012, 014 and 015 instructions. An attempt to execute these instructions to access ECS when the EA is clear shall cause an Error Exit (Illegal Instruction) to be executed in C170 State. (See table 7.2-2.)

EA shall be located in bit 4 of word 2 in the Exchange Package.

2.5.2.6 User Mask (UM)

UM shall be used by user processes to enable trap interrupts. There shall be 16 bits in the UM. See paragraph 2.8.4 for details.

The UM shall be located in bits 00 through 15 of word 3 in the Exchange Package.

2.5.2.7 Monitor Mask (MM)

MM shall be used by the monitor to enable exchange interrupts. There shall be 16 bits in the MM. See paragraph 2.8.2 for details.

The MM shall be located in bits 00 through 15 of word 4 in the Exchange Package.

2.5.2.8 User Condition Register (UCR)

UCR shall be a 16-bit register which records the occurrence of specified conditions within the processor. See paragraph 2.8.3 for details.

UCR shall be located in bits 00 through 15 of word 5 in the Exchange Package.

2.5.2.9 Monitor Condition Register (MCR)

MCR shall be a 16-bit register which records the occurrence of specified conditions within the processor and central memory. See paragraph 2.8.1 for details.

MCR shall be located in bits 00 through 15 of word 6 in the Exchange Package.

2.5.2.10 Debug Mask (DM)

The DM shall consist of two flag bits and five mask bits which control and condition the debug operations as described in paragraph 2.7.2 of this specification.

The DM bits shall be located in bits 09 through 15 of word 36 in the Exchange Package. The assignments are as follows:

Bit 09 : End of List Seen flag
Bit 10 : Debug Scan in Progress flag
Bit 11 : Data Read mask
Bit 12 : Data Write mask
Bit 13 : Instruction Fetch mask
Bit 14 : Branching Instruction mask
Bit 15 : Call Instruction mask

2.5.2.11 Keypoint Mask (KM)

KM shall consist of a 16-bit mask which is tested during the execution of a Keypoint instruction as specified in paragraph 2.6.1.7.

The KM shall be located in bits 00 through 15 of word 8 in the Exchange Package.

2.5.2.12 Keypoint Code (KC)

This process state register is no longer implemented. See 2.6.1.7 for the current keypoint implementation.

2.5.2.13 Process Interval Timer (PIT)

PIT shall be a 32-bit counter which a process shall use to determine time intervals. See paragraph 2.5.3.1 for details.

The PIT shall be located in bits 0 through 15 of words 11 and 12 in the Exchange Package. Word 11 shall contain the leftmost 16 bits of PIT.

2.5.2.14 Base Constant (BC)

The BC is intended to provide a means to communicate within the operating system. In this context, software shall have exclusive control over the contents of BC.

The BC shall be located in bits 00 through 15 of words 13 and 14 in the Exchange Package. Word 13 shall contain the leftmost 16 bits of BC.

2.5.2.15 Model-Dependent Flags (MDF)

MDF shall consist of 16 bits. MDF shall be processor model-dependent and shall be defined in the processor model-dependent specification.

MDF shall be located in bits 00 through 15 of word 15 in the Exchange Package.

2.5.2.16 Segment Table Length (STL)

STL, plus one, shall specify the number of 64-bit entries in the associated Segment Table. (See 2.5.2.18.)

It shall be used to verify that references to the Segment Table are actually within the defined Segment Table. STL shall be a positive 12-bit value. (See paragraph 3.3.)

The STL shall be located in bits 4 through 15 of word 16 in the Exchange Package.

2.5.2.17 Untranslatable Pointer (UTP)

When the processor sets MCR52, 54, 57 or 60 because of an exception detection, the processor shall also load the address which could not be translated into the UTP. (See 2.8.1, 2.8.7 and appendix I.) This occurs regardless of *C170 or C180,* Job or Monitor state. This address is always a PVA except for the following cases. When a program interruption occurs as a result of Monitor Condition Register bit 52 being set because of an Address Specification Error either on the Purge Buffer instruction (2.6.5.3) with K=0, 1, 8 or 9 or on the Load Page Table instruction, UTP will contain the SVA which was associated with the Address Specification Error. The processor shall only alter the UTP when MCR52, 54, 57 and/or 60 is being set due to detection of the associated exception.

When an Invalid Segment or Access Violation occurs, the UTP shall be loaded with the ring, segment, and byte number of the address causing the exception detection. For those cases where the byte number of the address is incremented or altered as part of the defined instruction execution, any of the values which are allowed by the instruction definition are acceptable for the UTP.

When an Address Specification Error occurs, the UTP shall be loaded with the ring, segment and byte number of the address causing the exception detection. For thoses cases where the byte number of the address is incremented or altered as part of the defined instruction execution, any of the values which are allowed by the instruction definition and which exhibit the Address Specification Error are acceptable for the UTP.

When a Page Table Search without Find occurs, the UTP shall be loaded with the ring, segment and byte number of the address causing the exception detection. For those cases where the byte number is incremented or altered as part of the defined instruction execution, any of the values which are allowed by the instruction definition and which reference the missing page are acceptable for the UTP.

Paragraph 2.8 describes the UTP definition when more than one of MCR bits 52, 54, 57 or 60 is set.

The UTP shall be located in bits 16 through 63 of word 34 in the Exchange Package.

2.5.2.18 Segment Table Address (STA)

STA shall be a real memory byte address that points to the first entry in the Segment Table. (See paragraph 3.3.) STA shall be interpreted as equal to 0 modulo 8. An STA which has bit 33 set causes undefined processor operation when this address is used. STA shall be located in bits 00 through 15 of words 34 and 35 of the Exchange Package. Word 34 shall contain the leftmost 16 bits of STA.

The processor shall ignore the state of bits 32, 61, 62 and 63 and operate as if these bits are zero. The writing of any value other than zero into bits 32, 61, 62 and 63 shall cause undefined operation with respect to the subsequent value of these bits. These bits (32, 61, 62 and 63) shall be zero when read assuming the previous write was zero as noted in 2.1.3.5b. (Also see table 2.5-1.)

2.5.2.19 Last Processor Identification (LPID)

LPID shall consist of the 8-bit Processor Identification from the last processor which executed the process defined by the Exchange Package. LPID shall be located in bits 08 through 15 of word 7 in the Exchange Package. See 2.5.1.8 of this specification.

2.5.2.20 Trap Enables (TE)

TE shall consist of a 2-bit field that determines how traps shall be enabled. The bits in TE shall be set by the "Copy from Xk per (Xj)" instruction (Op. 0F). Although the bits in TE can be cleared by the "Copy from Xk per (Xj)" instruction they shall normally be cleared by the hardware action described below. See section 2.8.6 for a description of the trap interrupt operation and section 2.8.10 for a description of flag states.

a. Trap Enable Flip-flop (TEF)

   TEF shall be the flip-flop which enables a trap interrupt operation to occur when it is set. It shall be set as described above and shall be cleared by hardware whenever a trap interrupt occurs.

   TEF shall be located in bit 14 of word 2 in the Exchange Package.

b. Trap Enabled Delay (TED)

   TED shall be a flip-flop which delays the enabling of trap interrupts until after the next Return instruction (Op. 04) is executed. The trap enable shall be inhibited as long as TED is set. The Return instruction clears TED. TED shall be set by the Copy instruction as just previously described.

   TED shall be located in bit 15 of word 2 in the Exchange Package.

2.5.2.21 Trap Pointer (TP)

TP shall consist of a PVA which points to a code base pointer in a binding section. The TP shall be used whenever a trap interrupt occurs. (See 2.8.6.)

The TP shall be located in bits 16 through 63 of word 35 in the Exchange Package.

2.5.2.22 Debug Index (DI)

DI shall consist of a 6-bit word-index into the debug list. It shall record where the debug list search must resume after a debug list find has been processed. (See 2.7.2.3.)

The DI shall be located in bits 00 through 05 of word 36 in the Exchange Package.

2.5.2.23 Debug List Pointer (DLP)

DLP shall consist of a PVA that points to the first entry in the debug list. (See 2.7.2.1.)

The DLP shall be located in bits 16 through 63 of word 36 in the Exchange Package.

2.5.2.24 Top of Stack (TOS)

Each TOS shall consist of a PVA that points to the top of its associated stack. There shall be an individual TOS pointer for each of the 15 rings. The TOS's shall be located in bits 16 through 63 of words 37 through 51 in the Exchange Package. The TOS for ring 1 shall be located in word 37, the TOS for ring 2 shall be located in word 38, etc.

2.5.2.25 Model-Dependent Word (MDW)

MDW shall consist of 64 bits, shall be processor model-dependent and shall be defined in the processor model-dependent specification. MDW shall be located in bits 00 through 63 of word 33 in the Exchange Package.

2.5.2.26 Virtual Machine Identifier (VMID)

The VMID shall consist of 4 bits and shall reflect the virtual machine capability to be exercised, as well as that most recently exercised, in the execution of the associated process. The VMID shall be located in bit positions 04 through 07 of word 1 in the Exchange Package.

2.5.2.27 Untranslatable Virtual Machine Identifier (UVMID)

The UVMID shall consist of 4 bits and shall reflect the virtual machine capability that was required by a process or procedure but was not included in the associated processor's Virtual Machine Capability List at the time an Exchange operation, a Call instruction or a Return instruction was executed. The UVMID shall be located in bit positions 12 through 15 of word 1 in the Exchange Package. Values of 0-15 for this 4-bit field shall correspond to bit positions 48-63 respectively of the Virtual Machine Capability List (VMCL); see subparagraph 2.5.1.12.

2.5.2.28 Largest Ring Number (LRN)

LRN shall consist of 4 bits and shall be equal in value to the largest ring number for which there is a corresponding TOS entry in the associated Exchange Package. (See 2.5.2.24.) Usage of the LRN shall be specified on a model-dependent basis. LRN shall be located in bits 12 through 15 of word 37 of the Exchange Package.

2.5.2.29 Sample Program Instruction (SPI) Identifier

The SPI identifier shall consist of 6 bits in word 0 of the exchange package which may be:

- written into the exchange package,
- copied to/from the P register as part of the exchange,
- read by the PP as part of the P register read, or
- read by the CPU by the Copy instruction.

2.5.3 Timers

The Process Interval Timer and the System Interval Timer shall be free-running timers to the extent that, upon reaching a count of zero and recording the corresponding condition in the User or Monitor Condition Register as described in subparagraphs 2.8.3.4 and 2.8.1.12 of this specification, respectively, decrement operations shall continue to occur at the 1Mhz rate.

2.5.3.1 Process Interval Timer

The Process Interval Timer (PIT) shall consist of a 32-bit counter that shall decrement once each microsecond. When it decrements to zero, it shall set the Process Interval Timer bit in the User Condition Register and continue to decrement from zero to FFFF FFFF and so on. When traps are enabled, execution of the current instruction shall complete and program interruption shall occur. See paragraph 2.8.3.4 of this specification.

The PIT contains a different count for each User process. When a particular process is not in active execution, its PIT value is stored in its Exchange Package. By this means, each User process may keep track of time intervals within its own program execution.

PIT shall be set by the "Copy from Xk per (Xj)" instruction described in section 2.6.5.2, as well as during an "Exchange" operation, described in 2.6.1.6 and 2.8.5.

An exchange operation (either to or from job mode) occuring when the PIT contains a value near zero shall not be allowed to cause the processor to miss setting the appropriate UCR bit when the PIT decrements to zero.

2.5.3.2 System Interval Timer

The System Interval Timer (SIT) shall consist of a 32-bit counter that shall decrement once each microsecond. When it decrements to zero, it shall set the System Interval Timer bit in the Monitor Condition Register and continue to decrement from zero to FFFF FFFF and so on. When the corresponding monitor mask bit is set, execution of the current instruction shall complete and program interruption shall occur as described in paragraph 2.8.1 of this specification.

By this means, the Monitor process may keep track of time intervals within the processor. SIT shall be set by the "Copy from Xk per (Xj)" instruction described in section 2.6.5.2.

2.5.4 Stacks

Each process shall have the means for addressing 15 stacks, one for each possible ring of execution as determined by the value of the ring number contained in the P Register.

The beginning of each stack shall be defined by the PVA referred to as the Top of Stack pointer, previously described in subparagraph 2.5.2.24 and illustrated in figure 2.5-1 of this specification.

Note: TOS pointers shall be addressed, using real addressing mode, as follows:

Address of TOS pointer = (Job Process State Register or Monitor Process State Register) plus (288) plus (8 times the value of the ring number contained in the P Register).

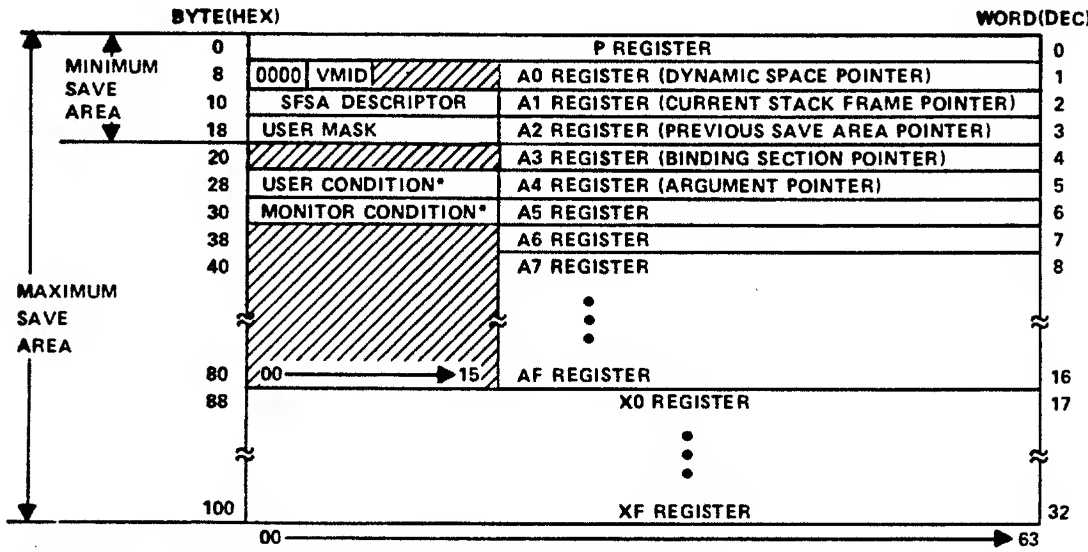
2.5.4.1 Stack Frames

Each stack shall be comprised of one or more stack frames. The beginning of each stack frame shall be defined by the PVA referred to as the Current Stack Frame Pointer. At the time a procedure is activated (or called) the CSF pointer shall be obtained by using the TOS pointer which corresponds to the procedure's ring of execution. During the time a procedure utilizes a stack frame, its length, from the beginning address, shall be defined as including each contiguous PVA up to, but not including, the PVA referred to as the Dynamic Space Pointer.

When within a process, a procedure "calls" another procedure, with the intention that the "called" procedure will "return" to its "caller," the stack frame associated with the "calling" procedure is intended to provide the means for preserving its environment so that its execution may be suspended, (at the time the other procedure is "called"), and then resumed, (at the time the "called" procedure "returns").

At the end of each stack frame, a "save area" shall be defined for that part of a procedure's "environment" which is implicit to the Call and Return instructions as defined in subparagraphs 2.6.1.2 through 2.6.1.4 of this specification. The stack frame save area shall consist of from four to thirty-three contiguous 64-bit words, beginning at the address defined by the Dynamic Space Pointer with respect to Call instructions and beginning at the address defined by the Previous Save Area Pointer with respect to the Return instruction.

The Stack Frame Save Area shall be formatted as follows:



* UCR AND MCR ARE STORED ON TRAP OPERATIONS. ON CALL OPERATIONS, UCR AND MCR POSITIONS IN THE SFSA ARE UNDEFINED.

Figure 2.5-1. Stack Frame Save Area (SFSA)

Thus, the "environment" which is implicit to the Call and Return instructions shall include:

Minimally;
- P Register
- Register A0 through A2 (DSP, CSF and PSA)
- Frame Description (SFSA Descriptor)
- User Mask
- Virtual Machine Identifier (VMID, see 2.5.6)

Selectively;
- Register A3 through AF (contiguously numbered)
- Register X0 through XF (contiguously numbered)

Notes: The PVA initially contained in the P Register shall be increased by four prior to writing the entire P Register (including its Global and Local Key fields), into the Current Stack Frame Save Area, Word 0, whenever such an operation occurs on the part of a Call instruction's execution.

Bits 0-3 of word 1 of the SFSA shall be written as zero in memory as the SFSA is written on a Call or Trap operation. These bits in the SFSA in memory are for use by applications and are not to be modified by an operating system

Unused fields in the SFSA may be changed to undefined values during the execution of Call instructions and shall be ignored during the execution of a Return instruction. This also includes bits 2 and 3 of the SFSA Descriptor.

The Stack Frame Save Area Descriptor shall consist of 16 bits formatted as follows:

00	01	02	03	04–07	08–11	12–15
CFF	OCF	//	PND	XS	AT	XT

CFF: Critical Frame Flag
OCF: On Condition Flag
PND: Process Not Damaged
XS: X Register, starting number (First X Reg. No.)
AT: A Register, terminating number (Last A Reg. No.)
XT: X Register, terminating number (Last X Reg. No.)

Bit 2 shall be ignored by the processor

Trap Interrupt shall generate a maximum Stack Frame Save Area (33 words), by definition.

For Call instructions, the A and X Registers to be stored into the Stack Frame Save Area shall be interpreted according to the contents of Register X0 Right, in the manner described in subparagraph 2.2.1.7 of this specification, with the exception that bit positions 48 through 51 of Register X0 Right shall be ignored and the storing of the A Register group shall unconditionally begin with Register A0. When X_S is greater than X_T, none of the X Registers shall be stored by Call instructions, and none shall be loaded by a Return instruction.

The execution of a Call instruction or a Trap Interrupt shall store the states of the Critical Frame and On Condition Flags into the Frame Descriptor associated with the Stack Frame Save Area. The execution of a Return instruction shall load these Flags from the Frame Descriptor contained within the previous Stack Frame Save Area.

The execution of a Trap Interrupt but NOT a Call instruction shall store the contents of the User Condition Register and Monitor Condition Register in bits 0-15 of words 5 and 6, respectively, of the Stack Frame Save Area. The bit or bits causing the trap shall then be cleared in the User and Monitor Condition Registers. That is, any bit set in a condition register, for which the corresponding bit is set in the appropriate mask register, shall be cleared. The execution of a Return instruction shall not restore the condition registers from the Stack Frame Save Area.

The execution of a Call Instruction or a Trap Interrupt shall store the Virtual Machine Identifier (VMID) associated with the "calling" or "trapped" procedure into bits 04 through 07 of Word 1 in the Stack Frame Save Area. The execution of a Return Instruction shall conditionally load bits 04 through 07 of Word 1 from the Previous Stack Frame Save Area to the VMID in the manner described in 2.5.6.

The PND, when set during a C180 Monitor Mode trap interrupt operation caused by an uncorrectable error, indicates that there is no evidence that the process being executed was damaged, and that it may be restarted. The PVA in P of the stack frame is the proper address to restart the process but is not necessarily the address of the instruction which initiated the activity that resulted in the malfunction. This flag is intended to allow recovery of monitor mode processes where possible.

Note that the default state of this flag is interpreted as process damaged. Thus, on a model-dependent basis, the hardware may detect many, few, or none of the undamaged processes and set PND accordingly. While a processor may report undamaged processes as damaged, it shall be a design objective to never report damaged processes as undamaged. Thus, a process denoted as undamaged via PND shall be as likely to be correct as any process with no fault indication. This flag shall be ignored by the hardware when loading a stack frame and is only defined in the stack frame resulting from a Detected Uncorrectable Error Interrupt in C180 monitor mode.

2.5.5 Binding Section Segment

A Binding Section Segment shall be identified by the RP field within its associated Segment Descriptor as described in 3.3.1.1 of this specification.

Binding Section Segments are intended to facilitate software linking of both code and data segments from one procedure to another.

2.5.5.1 Code Base Pointer

With respect to the Call instruction, as described in subparagraph 2.6.1.2 of this specification, having both inter-ring and inter-segment branching capabilities, a Binding Section Segment shall be used to contain the Code Base Pointer to the "called" procedure. The Code Base Pointer shall be located on a word boundary, shall consist of 64 bits and shall have the following format:

00	04	08	09	12	16	20	32 ... 63
///// /////	CBP-VMID	EPF	///// /////	CBP-R3	CBP-RN	SEG	BN
4	4	1	3	4	4	12	32

With respect to the "called" procedure these fields shall have the following interpretation:

CBP-VMID: Code Base Pointer, Virtual Machine Identifier
EPF: External Procedure Flag
CBP-R3: Code Base Pointer, Highest Ring Number for Call
CBP-RN: Code Base Pointer, Ring Number
SEG: Segment Number
BN: Byte Number

Bits 0-3 and 9-11 shall be ignored by the processor.

Note: When the External Procedure Flag is a one, the next contiguous word location from the Code Base Pointer shall contain a PVA in its rightmost 48-bit positions, 16 through 63, referred to as a Binding Section Pointer. Thus, a new Binding Section Pointer shall be provided (at the address of the Code Base Pointer plus 8) when an "external procedure" is "called."

2.5.6 Virtual Machine

Virtual Machine support shall involve the VMCL defined in 2.5.1.12, the UVMID defined in 2.5.2.27, as well as the 4-bit VMID fields from the Exchange Package defined in 2.5.2.26, the Stack Frame Save Area defined in 2.5.4.1, and the Code Base Pointer defined in 2.5.5.1, of this specification. Values of 0 through 15 for these VMID fields shall correspond to bit positions 48 through 63, respectively, of the VMCL. A match between VMID and VMCL shall exist whenever the corresponding bit position within the VMCL is a one.

a. Exchange operations shall include the check for VMID versus VMCL: when a match exists, these operations shall occur as defined in 2.8.5. When a mismatch exists, an Environment Specification Error shall be recorded along with the UVMID in the new (target) process' Exchange Package and the processor shall exchange, trap, or halt according to table 2.8-1 of this specification. (Note that the job-to-monitor or monitor-to-job transition shall occur regardless of the VMID/VMCL mismatch with respect to interpreting table 2.8-1.)

b. Call and Trap operations shall include the check for CBP-VMID versus VMCL: when a match exists, these operations shall occur as defined in 2.6.1.2 and 2.8.6, respectively, including the transfers of the initial contents of the VMID register to the Stack Frame Save Area, (Word 1, Byte 0) and the CBP-VMID field to the VMID Register final. When a mismatch exists, an Environment Specification Error shall be recorded along with the UVMID in the current process' Exchange Package and the processor shall exchange, trap or halt according to table 2.8-1 of this specification. (Note: A CBP-VMID/VMCL mismatch during a trap operation shall include the setting of the Trap Exception bit.)

c. The return instruction shall include the check of the VMID contained in Word 1, Byte 0 of the Stack Frame Save Area versus the VMCL: when a match exists, this operation shall occur as defined in 2.6.1.4 including the transfer of the VMID from the Stack Frame Save Area to the VMID Register associated with the current process. When a mismatch exists, an Environment Specification Error shall be recorded along with the UVMID in the current process' exchange package and the processor shall exchange, trap, or halt according to table 2.8-1 of this specification.

2.5.7 System Deadstart

The system deadstart procedure shall:

a. Set Monitor Mode (this shall set bit 58 of the Status Summary Register).
b. Load the Monitor Process exchange package from the address pointed to by the Monitor Process State Register.
c. Begin execution of the Monitor Process.

2.6 SYSTEM INSTRUCTIONS

Several of the system instructions require certain privileges as specified by the XP field in the segment descriptor (3.3.1) or require the processor to be in Monitor mode. The following chart is a summary of these requirements.

	Code Segment Attribute			Mode Requirement	
	Non-Privileged	Local Privileged	Global Privileged	Job	Monitor
All instructions other than those following	Execute	Execute	Execute	/ / /	/ / /
Interrupt Proc. (Op. 03)	Priv.Inst.Ft	Priv.Inst.Ft	Execute	/ / /	/ / /
Return (Op.04)				/ / /	/ / /
SFSA VMIO = 0	Execute	Execute	Execute	/ / /	/ / /
SFSA VMIO ≠ 0	Env.Spec.Err	Env.Spec.Err	Execute	/ / /	/ / /
Purge Buffer (op. 05)				/ / /	/ / /
K = 0, 1, 2, 8 → F	Priv.Inst.Ft	Execute	Execute	/ / /	/ / /
K = 3 → 7	Execute	Execute	Execute	/ / /	/ / /
Copy to Reg. (Op. 0E)					
Registers 00 → 5F	No-op	No-op	No-op	No-op	No-op
Registers 60 → 7F	/ / / /	/ / / /	/ / / /	Inst.Spec Error	Execute
Registers 80 → BF	Priv.Inst Ft	Priv.Inst.Ft	Execute	/ / /	/ / /
Registers C0 → 0F	Priv.Inst.Ft	Execute	Execute	/ / /	/ / /
Registers E0 → FF	Execute	Execute	Execute	/ / /	/ / /
Load Pg.Tbl.Index(Op.17)	Priv.Inst.Ft	Execute	Execute	/ / /	/ / /
Brch.on Cond.Reg.(Op.9F) K = 0, 1, 8, 9	/ / / /	/ / / /	/ / / /	Inst.Spec Error	Execute
K = 2 → 7, A → F	Execute	Execute	Execute	/ / /	/ / /

2.6.1 Nonprivileged System Instructions

The following system instructions shall be permitted to execute for any executable segment with the single exception described on the Return instruction (2.6.1.4).

2.6.1.1 Program Error

00jk (Ref. 121)

The execution of this instruction shall result in the detection of an Instruction Specification error and the corresponding program interruption shall occur. (See 2.8.1.4.)

The operation code for this instruction shall consist entirely of zeros.

The j and k fields from this instruction shall not be translated and their values shall have no effect on the execution of this instruction.

2.6.1.2 Call Indirect

Call per (Aj displaced by 8*Q), arguments per Ak

B5jkQ (Ref. 115)

Operation: This instruction shall save the environment (2.5.4.1) as designated by the contents of Register X0 Right, in the Stack Frame Save Area (SFSA) pointed to by the Dynamic Space Pointer initially contained in Register A0. The stack associated with the current ring of execution, as determined by the RN field initially contained in the P Register, shall be "pushed" by transferring the Dynamic Space Pointer, modified in its rightmost 32-bit positions by the addition of 8 times the number of words stored into the stack frame save area, to the appropriate Top of Stack entry in the executing process' Exchange Package. The PVA obtained from Register Aj shall be modified in its rightmost 32-bit positions by the addition of the sign-extended Q field from the instruction, (shifted left 3 bit positions with zeros inserted on the right), and the resulting PVA shall be used to address a Code Base Pointer from a Binding Section Segment. This Code Base Pointer shall be translated into a PVA used to address the first instruction to be executed in the "called" procedure. The ring of execution of the called procedure, P(RN) final, shall be used to obtain a Top of Stack pointer from the process' Exchange Package to be used as the new Current Stack Frame Pointer.

The processor may assume a 33-word SFSA when prevalidating this instruction. This has the effect of allowing a Page Table Search Without Find interrupt to occur at points where a SFSA actually terminates within a page but a maximum SFSA extends across the page boundary. This also has the effect of allowing an Address Specification Error interrupt to occur at points where a SFSA actually terminates with bit 32=0 but a maximum frame extends into the range where bit 32=1. The actual store into the SFSA during instruction execution shall only store into the SFSA as described by the Stack Frame Descriptor.

The processor may but need not prevalidate Aj+8*Q+8 for Address Specification Error and Page Table Search Without Find regardless of the state of the External Procedure Flag.

The A0, A1, and A2 Registers shall be altered to reflect changes with respect to the Current and Previous Stack Frames and the A3, and A4 Registers shall be altered to reflect pertinent parameter changes as required, in accomplishing this transfer of control from a "calling" procedure to a "called" procedure.

Register assignments shall be as follows:

(A0) - Dynamic Space Pointer
(A1) - Current Stack Frame Pointer
(A2) - Previous Save Area Pointer
(A3) - Binding Section Pointer
(A4) - Argument Pointer

Virtual machine support shall be provided by the execution of this instruction to the extent previously described in paragraph 2.5.6 of this specification.

For the purpose of referencing the 64-bit Code Base Pointer as previously described in subparagraph 2.5.5.1 of this specification, an Address Specification Error shall be recorded when the initial contents of Register Aj are not 0, modulo 8.

The associated program interruption shall occur as described in paragraph 2.8.1 of this specification, and the execution of this instruction shall be inhibited (except that portions of the environment may be stored into the SFSA and A0 may be rounded up before the instruction is inhibited) when any of the following exceptions are recorded.

Instruction Specification Error (See 2.5.4.1 and 2.8.1.4)

Value of the four bits in bit positions 56 through 59 of X0 Right is less than 2.

Environment Specification Error (See 2.8.1.8 and 2.5.6)

Code Base Pointer VMID mismatch with VMCL.

Outward Call/Inward Return (See 2.8.1.14)

Initial P Ring Number less than Segment Descriptor R1.

Invalid Segment
Access Violation
Address Specification Error
Page Table Search Without Find

See Appendix I for these four exception conditions.

Note: Steps a and b of the following execution sequence may occur out of order in relation to steps c through s insofar as the rounding of Register A0 and the storing of the "Environment" into the SFSA in central memory, (including the associated exception sensing), are concerned.

In the absence of a program interruption, the following sequence of events shall accomplish the execution of the instruction:

	Operation	Remarks
* a.	(A0) + n − 1 to A0, 0 modulo n result ‡	Round DSP upward
* b.	"Environment" to Stack Frame Save Area	See paragraph 2.5.4.1
* c.	Copy P Left to X0 Left	Copy Caller's ID
* d.	Store (A0), all 48 bits, incremented by 8 times the number of save area words, to Top of Stack pointer for current ring number	Update TOS pointer See paragraph 2.5.2.24
* e.	Load P Key with Segment Descriptor Lock for callee	
* f.	If P Ring Number is less than callee Segment Descriptor R2, go to step h	Intra-ring Call
* g.	Set P Ring Number equal to callee Segment Descriptor R2	Inward Call
* h.	Load P SEG and BN fields with Code Base Pointer SEG and BN fields	
* i.	If CBP-VMID≠0, go to step m	Test destination machine CYBER 180
* j.	If Code Base Pointer EPF is 0, go to step l	Internal Procedure
* k.	Load A3 with new Binding Section Pointer	Ring Number stored into A3 shall be the larger of the ring number in the BSP from caller's Binding Section and the new P ring number. See paragraphs 2.5.5.1and 3.2.1.1.
† l.	Copy (Ak) to A4	Pass parameters. When k is 0-3, the final contents of A4 shall be undefined with respect to which A register is transferred into A4.
* m.	Copy (A0) to A2	DSP from step a to PSA pointer
* n.	Clear On Condition Flag	Clear OCF
* o.	Load A1 with new Top of Stack pointer per final P Ring Number and clear Critical Frame Flag	TOS to CSF pointer Clear CFF
* p.	Copy (A1) to A0	CSF pointer to DSP
* q.	Copy CBP-VMID to VMID Register	See 2.5.6

* Unconditionally included in a Trap Interrupt
† Unconditionally omitted from a Trap Interrupt (see 2.8.6)
‡ In step a, n is any multiple of 8 greater than zero and less than 272.

2.6.1.3 Call Relative

Call to P displaced by 8*Q, Binding Section Pointer per Aj, arguments per Ak

B0jkQ (Ref. 116)

Operation: This instruction shall save the "environment," as designated by the contents of Register X0 Right, in the Stack Frame Save Area (SFSA) pointed to by the Dynamic Space Pointer initially contained in Register A0. The stack associated with the current ring of execution, as determined by the RN field initially contained in the P Register, shall be "pushed" by transferring the Dynamic Space Pointer, modified in its rightmost 32 bit positions by the addition of 8 times the number of words stored into the stack frame save area, to the appropriate Top of Stack entry in the executing process' Exchange Package.

The P Register shall be modified in its rightmost 32 bit positions by the sign extended Q field from the instruction, (left-shifted 3 bit positions with zeros inserted on the right). The final contents of the P Register shall be made 0, modulo 8, by clearing the least significant 3 bit positions (61-63) and shall be used to address the first instruction to be executed in the "called" procedure.

The processor may assume a 33-word SFSA when prevalidating this instruction. This has the effect of allowing a Page Table Search Without Find interrupt to occur at points where a SFSA actually terminates within a page but a maximum SFSA extends across the page boundary. This also has the effect of allowing an Address Specification Error interrupt to occur at points where a SFSA actually terminates with bit 32=0 but a maximum frame extends into the range where bit 32=1. The actual store into the SFSA during instruction execution shall only store into the SFSA as described by the Stack Frame Save Area Descriptor.

Registers A0, A1 and A2 shall be altered to reflect changes with respect to the Current and Previous Stack Frames and the A3 and A4 Registers shall be altered to reflect pertinent parameter changes as required, in accomplishing this intra-ring, intra-segment transfer of control from a "calling" procedure to a "called" procedure.

Register assignments shall be as follows:

(A0) - Dynamic Space Pointer
(A1) - Current Stack Frame Pointer
(A2) - Previous Save Area Pointer
(A3) - Binding Section Pointer
(A4) - Argument Pointer

Note: Steps a and b of the following execution sequence may occur out of order in relation to steps c through i insofar as the rounding of Register A0 and the storing of the "environment" into the CSF save area in central memory are concerned.

The associated program interruption shall occur as described in paragraph 2.8.1 of this specification, and the execution of this instruction shall be inhibited (except that portions of the environment may be stored into the Stack Frame Save Area and A0 may be rounded up before the instruction is inhibited) when any of the following exceptions are recorded:

Instruction Specification Error when the value of the 4 bits in positions 56 through 59 of Register X0 Right is less than 2.

Invalid Segment
Access Violation
Address Specification Error
Page Table Search Without Find

See Appendix I for these four exception conditions.

In the absence of a program interruption, the following sequence of events shall accomplish the execution of this instruction:

	Operation	Remarks
a.	(A0) + n - 1 to A0, 0 modulo n result*	Round DSP upward
b.	"Environment" to Stack Frame Save Area	See paragraph 2.5.4.1
c.	Copy P Left to X0 Left	Copy Caller's ID
d.	Store rounded (A0), incremented by 8 times the number of save area words, to Exchange Package per initial P Ring Number	Update TOS pointer See paragraph 2.5.2.24
e.	(P) plus 8*Q, 0 modulo 8, to P	Intra-ring, Intra-segment Call
f.	Copy (Aj) to A3 and (Ak) to A4	Pass Parameters. When k is 0-3, the final contents of A4 shall be undefined with respect to which A register is transferred into A4. When j is 0-2, the final contents of A3 shall be undefined with respect to which A register is transferred into A3.
g.	Copy rounded (A0) to A2 and clear Critical Frame Flag	DSP from step a to PSA pointer. Clear Flag.
h.	Copy rounded (A0), incremented by 8 times the number of save area words, to A0 and A1	TOS to CSF pointer and DSP
i.	Clear On Condition Flag	Clear OCF

* In step a, n is any multiple of 8 greater than zero and less than 272.

2.6.1.4 Return

04jk (Ref. 117)

Operation: This instruction shall reestablish the Stack Frame and "environment" of a previous procedure as defined by the Previous Save Area Pointer.

The j and k fields from this instruction shall not be translated. Thus, their values shall have no effect on the execution of this instruction for which all execution parameters shall be implicit.

The Stack Frame Save Area from which a previous procedure's "environment" shall be obtained, shall be addressed by means of the PVA initially contained in Register A2. (See 2.9.4 for SFSA Pushdown description.) The format of the previous procedure's Stack Frame Save Area shall conform to the description contained in paragraph 2.5.4.1 of this specification. This operation of loading the environment does not include loading or altering either MCR or UCR.

Virtual machine support shall be provided by the execution of this instruction to the extent previously described in paragraph 2.5.6 of this specification.

The processor may assume a 33-word Stack Frame Save Area when prevalidating the previous SFSA. This has the effect of allowing a Page Fault interrupt to occur at points where an SFSA actually terminates within a page but a maximum frame extends across the page boundary. This also has the effect of allowing an Address Specification Error interrupt to occur at points where an SFSA actually terminates below $2^{32}=1$ but a maximum frame extends beyond $2^{32}=1$. The actual load of the SFSA during instruction execution shall only load the SFSA as described by the SFSA Descriptor.

The associated program interruption shall occur as described in paragraph 2.8.1 of this specification, and the execution of this instruction shall be inhibited or completed as specified when any of the following exceptions are recorded:

Environment Specification Error

The value of the field designating the last A Register to be loaded, as contained in the previous Stack Frame Save Area Descriptor, is less than 2.*

Final (A0) would not equal initial (A2). This test may but need not be implemented.

Previous Save Area VMID mismatch with VMCL is per paragraph 2.5.6.†

Attempt to execute a Return instruction not having Global Privilege to a Previous Stack Frame Save Area containing a VMID≠0.

Attempt to execute a Return instruction with some data in the SFSA Pushdown but the SFSA is not accessible in central memory. In this case, the processor may but need not also report the exception condition which was encountered. If an Address Spec Error, Page Table Search Without Find, Invalid Segment, or Access Violation is reported along with the Environment Spec Error, the UTP must be appropriately loaded. Detection of an Environment Spec Error because the SFSA is not accessible may cause the hardware to leave garbage in the A and X registers and in the TOS pointer for the target ring. The hardware (including the Service Processor) shall leave the P of the process to which control was being passed in the P register and shall not leave an A register with a ring number less than the P ring number.

Outward Call (Inward Return) if final P ring number would be less than initial A2 ring number.‡

* This test is not required when the SFSA Descriptor is retrieved from the SFSA Pushdown.
† This test is not required when the VMID is retrieved from the SFSA Pushdown.
‡ This test is not required when the P ring number is retrieved from the SFSA Pushdown.

Critical Frame Flag if the initial state of the Critical Frame Flag is equal to a one.

Invalid Segment
Access Violation
Address Specification Error
Page Table Search Without Find

See Appendix I for these four exception conditions.

In the absence of a program interruption, the following sequence of events shall accomplish the execution of this instruction:

Operation:

a. Load the "environment" from the Stack Frame Save Area (SFSA) pointed to by the PVA initially contained in Register A2. The "environment" consists of the following:

- P register (64 bits) including Key.

  This results in a Branch exit for the RETURN instruction (2.1.3.4).

- VMID (4 bits)
- Critical Frame Flag and On Condition Flag to be set/cleared as per the Stack Frame Descriptor (2.5.4.1).
- User Mask (16 bits)
- Registers A0 through AT as specified in the SFSA Descriptor. As part of this load of the A register, the larger value of the following shall be transferred to the Ring Number (bits 16-19) of each A register:
  1) the ring number of the A register as obtained from the Stack Frame Save Area.
  2) the initial ring number of the A2 register.
  3) the R1 field contained in the segment descriptor associated with the initial A2.
- X registers (64 bits each) as per the SFSA Descriptor.

b. The ring number of the A registers not loaded in step a shall be unaltered by this instruction <u>except</u> when step a causes the P ring number to change value. When this occurs each A register not loaded in step a and having a ring number less than the new P ring number shall have its ring number set equal to the new P register ring number.

c. Store the final (A1) to the Exchange Package per the final P ring number | CSF → TOS pointer. (See paragraph 2.5.2.24.)

d. Clear the Trap Enable Delay. (See 2.5.2.20.)

Ring Number Zero Test: [See section 1.6 for a list of the C180 models that do not perform this test.] A Ring Number Zero condition shall be recorded when the RN=0 for any A register read from the previous Stack Frame Save Area. A Ring Number Zero condition shall not inhibit the execution of the Return instruction nor shall anything be placed in the UTP Register as a result of the Ring Number Zero.

2.6.1.5 Pop

06jk (Ref. 118)

Operation: This instruction shall reestablish the Stack Frame of a previous procedure as defined by the previous Stack Frame Save Area.

The j and k fields from this instruction shall not be translated. Thus, their values shall have no effect on the execution of this instruction for which all execution parameters shall be implicit.

The Stack Frame Save Area from which a previous procedure's Stack Frame pointers shall be obtained, shall be addressed by means of the PVA initially contained in Register A2. (See 2.9.4 for SFSA Pushdown description.) The format of the previous procedure's Stack Frame Save Area shall conform to the description contained in paragraph 2.5.4.1 of this specification.

The Pop instruction on processors having a SFSA Pushdown may:

- use the top SFSA in the SFSA Pushdown and pop the SFSA off of the SFSA Pushdown, or
- use the SFSA from central memory and pop a SFSA off of the SFSA Pushdown, or
- use the SFSA from central memory and purge the SFSA Pushdown.

The associated program interruption shall occur as described in paragraph 2.8.1 of this specification, and the execution of this instruction shall be inhibited when any of the following exceptions are recorded:

Environment Specification Error

Initial (A2) not equal to Word 1 contained in the previous procedure's Stack Frame Save Area. This test may but need not be implemented.

Inter-Ring Pop if the RN field contained in the P Register is not equal to the RN field initially contained in Register A2.

Critical Frame Flag if the initial state of the Critical Frame Flag is equal to a one.

Invalid Segment
Access Violation
Address Specification Error
Page Table Search Without Find

See Appendix I for these four exception conditions.

In the absence of a program interruption, the following sequence of events shall accomplish the execution of this instruction:

Operation	Remarks
a. Load A1 with the PVA from Word 2 of the previous procedure's Stack Frame Save Area. Set A1.RN equal to the largest of: • the ring number of A2 at the beginning of the POP instruction, • the ring number of A1 as read from the SFSA, or • the R1 field from the segment descriptor entry for the segment associated with the PVA used to obtain the new A1.	Update CSF Pointer
b. Load A2 with the PVA from Word 3 of the previous procedure's Stack Frame Save Area. Set A2.RN equal to the largest of: • the ring number of A2 at the beginning of the POP instruction, • the ring number of A2 as read from the SFSA, or • the R1 field from the segment descriptor entry for the segment associated with the PVA used to obtain the new A2.	Update PSA Pointer
c. Load the Critical Frame Flag and the On Condition Flag from the previous procedure's Stack Frame Save Area.	Update CFF and OCF
d. Store the final (A1) to the process' Exchange Package per the P Register's Ring Number	Update TOS Pointer

Ring Number Zero Test: [See section 1.6 for a list of the C180 models that do not perform this test.] A Ring Number Zero condition shall be recorded when the RN=0 for A1 or A2 as read from the previous Stack Frame Save Area. A Ring Number Zero condition shall not inhibit the execution of the POP instruction nor shall anything be placed in the UTP register as a result of the Ring Number Zero.

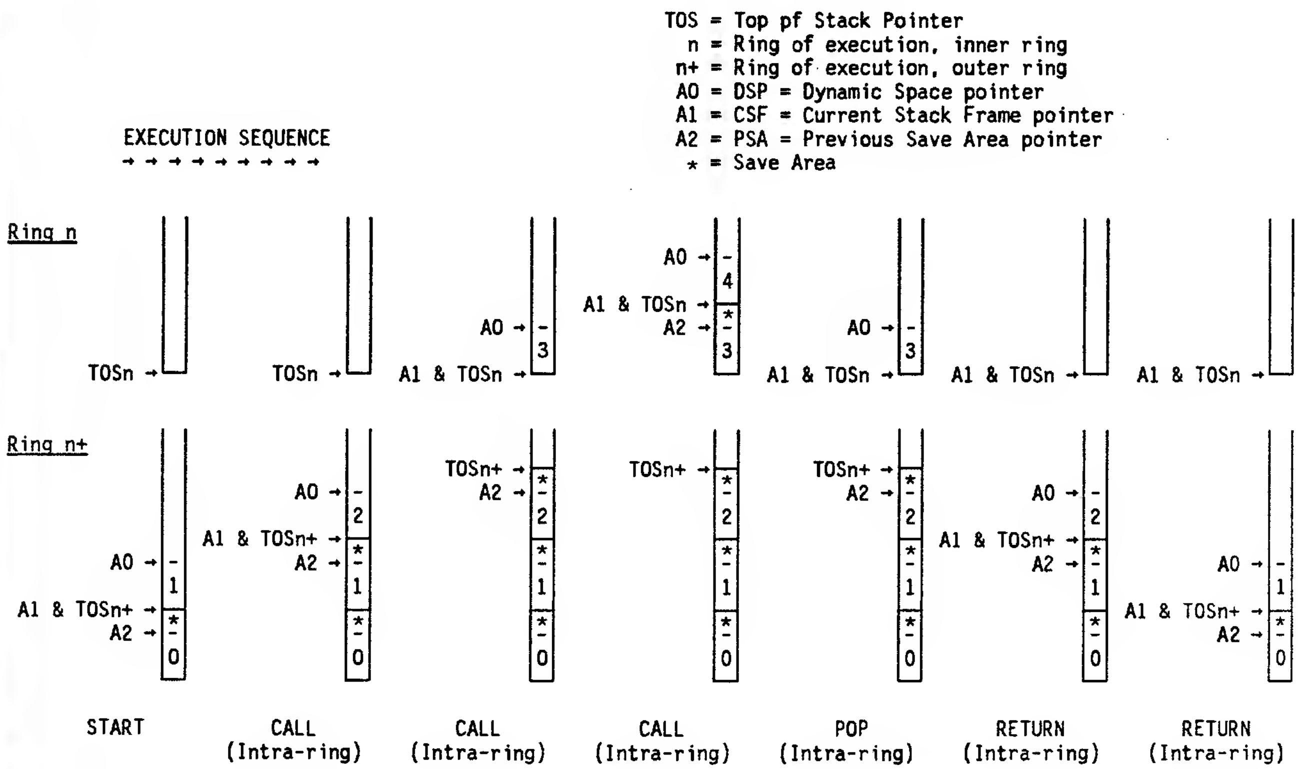


Figure 2.6-1. Call/Return/Pop, Post-execution Stack Frame States (including the software updating the Dynamic Space Pointer)

2.6.1.6 Exchange

02jk (Ref. 120)

When executed in Monitor mode this instruction shall change the processor from monitor process state to job process state. (See 2.8.5.2.)

When executed in Job mode this instruction shall change the processor from job process state to monitor process state. (See 2.8.5.1.) In addition, the System Call bit in position 58 of the Monitor Condition Register, job process state, shall be set.

The only exception condition generated by the execution of an Exchange operation is an Environmental Specification Error as described in 2.5.6. The Exchange is allowed to complete, the Environmental Specification Error is recorded in the Monitor Condition Register and the Monitor and User Condition Registers are examined as described in 2.8.5, 2.8.7, and 2.8.8.

The PVA contained in Word 0 (P Register) of the Exchange Package associated with the state from which the exchange is taking place, shall be updated such that it points to the instruction which would have been executed had the exchange not taken place, i.e., the PVA of the "Exchange" instruction with 2 added to its BN field.

The j and k fields from this instruction shall not be translated and their values shall have no effect on the execution of this instruction.

2.6.1.7 Keypoint

Keypoint, class j, code equal to XkR plus Q

B1jkQ (Ref. 136)

The Keypoint instruction allows program monitoring via the storage of a Keypoint data word or entry into central memory. The j field, termed the Keypoint Class Number (KCN), shall be used as a bit index into the Keypoint Mask Register (KMR). Thus, a KCN or j field having the value of 4 tests the 5th bit from the left in the KMR. The instruction shall only produce a Keypoint entry when both the Keypoint Enable Flag from the exchange package is set and the jth bit in the KMR is set.

The Keypoint entry is defind as follows:

00 Timer (28)	28 KCN (4)	32 Keypoint Code (32) 63

The Keypoint Code (32 bits) shall be formed by the addition of Q, expanded to 32 bits by means of sign extension, to the contents of Register Xk Right. For the purpose of this instruction, the contents of Register X0 Right shall be interpreted as consisting of zeros. Arithmetic overflow shall not be detected during the formation of the Keypoint Code. The 28 bits of the timer shall be equivalent to the rightmost 28 bits of the Free Running Counter in central memory. This 28-bit field may either be taken directly from the Free Running Counter or from a separate counter implemented within the processor. When implemented as a separate counter within the processor, the counter shall be loaded from the Free Running Counter as part of initialization of the processor. This requirement of being in "sync" with the Free Running Counter is to ensure that systems having more than one processor shall produce keypoint entries having time correlation among the processors. The 28-bit microsecond timer will cycle every 268 seconds.

The Keypoint entry as described above shall be written into central memory at the location specified by the PVA contained in the rightmost 48 bits of the Keypoint Buffer Pointer (KBP). After writing the Keypoint data into central memory, the processor shall increment the rightmost 32 bits of the PVA in the KBP by 8(hex). The rightmost 3 bits of the KBP shall be ignored.

No exception conditions associated with the KBP shall be reported unless the keypoint conditions are all met such that a write operation is required.

Invalid Segment
Access Violation
Address Specification Error
Page Table Search Without Find

See Appendix I for these four exception conditions.

2.6.1.8 Compare Swap

Compare Xk to (Aj); if locked, branch to P displaced by 2*Q; if unlocked, load/store (Aj), result to X1R

B4jkQ (Ref. 125)

When the 64-bit word in central memory, whose PVA is contained in Register Aj, initially consists entirely of ones in the leftmost 32-bit positions, i.e., locked, this instruction shall perform a branch exit in the manner previously described under the heading "branch exit" in paragraph 2.2.3 of this specification. In the absence of such a condition, this instruction shall perform a normal exit upon completion of the following operations. The 64-bit word initially contained in Register Xk shall be compared (64-bit integer compare) to the interlock word in central memory whose PVA is contained in Register Aj, and if equality is found, the contents of Register X0 shall be stored in central memory at the PVA contained in Register Aj, and Register X1 Right shall be cleared in all 32-bit positions. If equality is not found, Register Xk shall be loaded with the central memory word whose PVA is contained in Register Aj. When the initial contents of Register Xk are greater than the central memory word whose PVA is contained in Register Aj, Register X1 Right shall be cleared in bit positions 32 and 34 - 63, and set in bit position 33. When the initial contents of Register Xk are less than the central memory word whose PVA is contained in Register Aj, Register X1 Right shall be cleared in bit positions 34 through 63, and shall be set in bit positions 32 and 33. The final contents of Register X1 Right shall be undefined when k=1.

A serialization function shall be performed before this instruction begins and again at its ending. Execution of this instruction shall be delayed until all previous accesses to central memory on the part of this processor are completed. Execution of subsequent instructions shall be delayed until all central memory accesses due to this instruction are completed.

Conceptually, the execution of this "Compare" instruction on the part of a processor shall result in preventing other processors from any port from altering or transferring to an X register the central memory word at the PVA contained in Register Aj between the read and write accesses associated with the execution of this instruction, provided such processors are also executing a "Compare" instruction. With respect to this instruction only, in order to satisfy its "nonpreemptive" requirement, the use of 64-bit words consisting entirely of ones in their leftmost 32-bit positions, 00 through 31, shall be reserved for each processor's implementation of this instruction. When the 32-bit halfword initially contained in Register X0 Left consists entirely of ones, an Instruction Specification Error shall be detected, the execution of this instruction shall be inhibited, and the corresponding program interruption shall occur.

Notes: The tests for all ones in the left half of the word obtained from central memory and the word contained in X0 allow the following hardware implementation. The initial read of central memory is performed with an Exchange function (4.2) which both obtains the word from central memory and sets the left half to all ones in central memory. Before exiting, this implementation must either restore the initial contents of central memory or store X0 into central memory. (Note that on the branch exit, the left half was initially all ones, thus, no second store is required because the word was not altered.) This implementation exerts its non-preemptive requirement by using the exchange function to store all ones, thus, forcing a second processor to take the branch exit if initiating a compare instruction after the first processor's initial reference and before the first processor completes the instruction.

The hardware is not limited to this implementation to achieve the nonpreemptive requirement but must perform the two tests for all ones to support the processors which do use this approach.

For the purpose of establishing operand access validation, the central memory operand access types shall consist of both a read and a write access. Moreover, those processors having a Cache shall bypass it with respect to the read access and shall purge the associated entry from it with respect to the write access, (see 2.9). (No purge of cache is required on the branch exit.) Unless the central memory operand address consists of a byte address which is 0, modulo 8, an Address Specification error shall be detected, the execution of this instruction shall be inhibited, and the corresponding interruption shall occur.

With respect to Debug Scan operations as described in paragraph 2.7.2 of this specification, the Compare and Swap instruction, (Op. B4), shall assume the operands are not "locked," that the addresses of the operands shall be used for both read and write reference arguments for the purpose of scanning the Debug List and that the branch addresses, to be used when the operands are "locked" shall not be used as arguments for the scanning of the Debug List.

2.6.1.9 Test and Set Bit

Load bit to XkR from (Aj bit indexed by X0R), and set bit in central memory

14jk (Ref. 124)

Operation: - This instruction shall transfer a single bit into Register Xk Right, bit position 63, from a bit position in central memory. This instruction shall also clear the Xk Register in its leftmost 63-bit positions, 00 through 62. The bit position in central memory shall be unconditionally set without changing any other bit positions within the byte or word.

No other accesses from any port shall be permitted to the byte in central memory from the beginning of the read access until the end of the write access which sets the bit within that byte.

A serialization function shall be performed before this instruction begins and again at its ending. Execution of this instruction shall be delayed until all previous accesses to central memory by this processor are completed. Execution of subsequent instructions by this processor shall be delayed until all central memory accesses from this instruction are completed.

Addressing: - The byte in central memory, containing the bit position to be loaded shall be addressed by means of the PVA contained in the Aj Register modified by a bit item count, consisting of a 32-bit index, as follows: the 32-bit halfword obtained from Register X0 Right shall be shifted right three bit positions, end-off with sign extension on the left, and the 32-bit shifted result shall be added to the rightmost 32 bits of the PVA obtained from the Aj Register.

Bit Select: - The bit position within the addressed byte in central memory shall be selected by means of the rightmost three bits obtained from Register X0 Right, bit positions 61 through 63. Values from 0 through 7 for these three bits shall select the corresponding bit position, 0 through 7 from the central memory byte.

Notes: For the purpose of establishing access validity, the central memory operand access types shall consist of a read and a write access. Moreover, those processors having a Cache (see 2.9) shall bypass it with respect to the read access and shall purge the associated entry from it with respect to the write access.

2.6.1.10 Test and Set Page

Test Page (Aj) and set XkR

16jk (Ref. 126)

This instruction shall test for the presence of the page in central memory corresponding to the PVA contained in Register Aj. The test of the Page Table Entries includes testing that the valid bit is set for the Page Table Entry that satisfies this search. The search may, but need not, be halted when a cleared continue bit is encountered in the Page Table (as described in 3.5.1.1). When this instruction finds the corresponding page in central memory, the "Used" bit in the UM field of the associated Page Descriptor shall be set, (see 2.9.1 and 3.5.1.1), and the Real Memory Address (RMA) translated from the PVA contained in Register Aj shall be transferred to Register Xk Right. When this instruction cannot find the corresponding page in central memory, Register Xk Right shall be set in bit position 32 and cleared in bit positions 33 through 63.

Notes: With respect to the PVA contained in Register Aj, Access Violation and Page Table Search without Find conditions, described in 2.8.1.7 and 2.8.1.10, shall be excluded and Address Specification Error (bit 32=1) and Invalid Segment conditions, described in 2.8.1.5 and 2.8.1.13 shall be included in the execution of this instruction insofar as Virtual Memory mechanism exception sensing is concerned. The PVA contained in Register Aj shall not be used as an argument for Debug Scan operations.

For those processors with a MAP buffer, this instruction need not cause any entry into the MAP to be made.

2.6.1.11 Copy Free Running Counter

Copy Free Running Counter to Xk at XjR

08jk (Ref. 132)

This instruction shall copy the Free Running Counter from central memory as specified by the contents of Register Xj into the Xk Register. All 64 bits of the Xk Register shall be cleared before the Free Running Counter is copied into it.

The Processor Memory Port to be utilized during the execution of this instruction shall be determined in the manner defined in subparagraph 2.10.1.1 of this specification with the exception that, for this instruction, bit 33 of the Xj Register shall be used in place of bit 33 of the Real Memory Address as described in that subparagraph, item a, and in paragraph 2.5.1.11. The remaining bits (32, 34-63) in Xj Right shall be zeros or else the operation of this instruction is undefined.

2.6.1.12 Execute Algorithm

CSjkiD (Ref. 139)

This instruction shall be a processor model-dependent instruction. As such, it shall be defined in the appropriate processor model-dependent specification.

Bit 0 of the 8-bit S field is always reserved in all models.

Note: For those processors in which one or more of the algorithms have not been implemented, the corresponding Execute Algorithm instructions shall result in the recording of an Unimplemented Instruction condition. (See 2.8.3.2.)

2.6.1.13 Unimplemented Instructions - Reserved Op Codes

BDjkQ (Ref. 202)

BEjkQ (Ref. 170)

BFjkQ (Ref. 171)

The unimplemented instruction BD is reserved for the Debug software to implement instruction fetch debug via the unimplemented instruction trap. The operation code for this instruction is reserved and will not be used in future hardware extensions.

The two unimplemented instructions BE and BF are reserved for software simulation of operations not provided in C180 via the trap mechanism. The operation codes for these instructions are reserved and will not be used in future hardware extensions.

2.6.1.14 Scope Loop Sync

01jk (Ref. 194)

The execution of this instruction shall result in both of the following:

- The hardware shall provide at a test point, a signal suitable for the synchronization of test equipment.
- Processors whose local central memory contains a refresh counter shall issue the Refresh Counter Resync function (4.2.3.5) to the local central memory (RMA bit 33 clear, 2.10.1.1) followed by a read of word 0 from the current C180 Exchange Package (via JPS or MPS).

The j and k fields from this instruction shall not be translated and their values shall have no effect on the execution of this instruction.

2.6.1.15 Purge SFSA Pushdown

07jk (Ref. 203)

SFSA Pushdown is a model-dependent feature. See section 1.6 for a list of the C180 systems supporting SFSA Pushdown.

For processors having a SFSA Pushdown, this instruction, when the sub-op k=1, shall cause the processor to purge the SFSA Pushdown as described in section 2.9.4.1. When k≠1, this instruction shall be executed as a no-op.

For processors not having a SFSA Pushdown this instruction shall be executed as a no-op for all values of k.

2.6.2 Local Privileged Mode

This class of instructions shall be permitted to execute only from segments having either local privileged mode or global privileged mode.

Instructions in the local privileged mode class shall be executable whenever a processor is executing instructions from a segment whose Segment Descriptor defines that segment as either a local privileged executable segment or a global privileged executable segment. (See 3.3.1.1.)

Local privilege is required for the Load Page Table Index instruction described below and for certain cases of the Copy to State Register instruction (2.6.5.2) and the Purge Buffer instruction (2.6.5.3).

2.6.2.1 Load Page Table Index

Load Page Table Index per Xj to XkR and set X1R

17jk (Ref. 127)

This local privileged instruction shall search the Page Table in central memory, shall return the final index value to Register Xk Right, and shall set Register X1 Right according to the results of the search.

The entry searched for within the Page Table shall be defined by the System Virtual Address (SVA) contained in Register Xj. For a description of the format for an SVA in an X Register, see subparagraph 2.6.5.3 of this specification.

The Page Table shall be searched in the manner normally employed by the Virtual Addressing Mechanism except that:

- The search is strictly sequential, and halted by a cleared Continue bit,
- Valid bits shall be ignored,
- The Page Map entry shall not be loaded into the Page Map if present (see 2.9.1),
- The Used bit shall not be altered in the Page Table entry.

Thus, the SVA shall be psuedo-randomized (hashed), in conjunction with the Page Table Length (PTL), in order to obtain a nominal index value in the manner described in subparagraph 3.5.2.1 of this specification. The Page Table Address (PTA) interpreted as equal to 0, modulo the PTL, shall be concatenated to this nominal index value for the purpose of determining the first location to be searched in the Page Table.

Beginning with this location, the Page Table shall be linearly searched, (with the nominal index value increased by 8 for each entry which does not correspond to the SVA but does contain a Continue bit equal to 1, up to a maximum of 32 entries searched) in the manner described in subparagraph 3.5.2.2 of this specification.

The number of entries searched shall always be transferred to Register X1 Right, bits 33-63, right-justified with zeros extended.

When a Page Descriptor corresponding to the SVA initially contained in Register Xj is found, the index (20-bit quantity as described in step 3 of paragraph 3.5.2.1 plus potential increases as described above) into the Page Table which is associated with that entry shall be transferred right-justified and zero-extended to Register Xk Right, and bit 32 of Register X1 Right shall be set. For those processors with a MAP buffer, this instruction shall not cause any entry into the MAP to be made other than the valid entry associated with the fetch of this instruction.

When the Page Table search terminates as a result of not finding a Page Descriptor which corresponds to the SVA initially contained in Register Xj, (whether the termination results from a Continue bit equal to 0 or performing a maximum of 32 comparisons), the index into the Page Table associated with the last entry compared shall be transferred into Register Xk Right and bit 32 of Register X1 Right shall be cleared.

When k is equal to 1, the final value in X1 shall be as defined above for X1 rather than as defined for Xk. With respect to the SVA contained in Register Xj, Access Violation, Page Table Search without Find and Invalid Segment, described in paragraphs 2.8.1.7, 2.8.1.10 and 2.8.1.13, shall be excluded and Address Specification Error (bit 32=1), described in paragraph 2.8.1.5, shall be included in the execution of this instruction insofar as Virtual Memory mechanism exception sensing is concerned.

When the instruction attempts execution from a segment having neither local nor global privileges, a Privileged Instruction Fault shall be detected, execution of that instruction shall be inhibited, and the corresponding program interruption shall occur.

2.6.3 Global Privileged Mode

This class of instructions shall be permitted to execute only from segments having global privileged mode.

Global privileged mode shall exist whenever the processor is executing instructions from a segment whose Segment Descriptor defines that segment as a global privileged executable segment. (See 3.3.1.1.)

Global privilege is required for the Interrupt Processor instruction described below and for certain cases of the Return instruction (2.6.1.4) and the Copy to State Register instruction (2.6.5.2).

2.6.3.1 Processor Interrupt

Processor Interrupt per Xk

03jk (Ref. 122)

The execution of this global privileged class instruction shall send an external interrupt to one or more processors via their central memory ports. The processors shall be identified by the central memory port number to which they are connected.

The interrupting processor shall send the contents of Register Xk to central memory. Central memory shall then send an external interrupt to the processor(s) on those ports corresponding to the bit positions set within Register Xk.

Xk Bit Number:	60	61	62	63
Port Number:	3	2	1	0

Bits 0-32, and 34-59 shall not be used to send interrupts, but shall be ignored (except that correct parity is required).

The Processor port (Local or External) to be utilized during the execution of this instruction shall be determined in the manner defined in subparagraph 2.10.1.1 of this specification with the exception that, for this instruction, bit 33 of the Xk Register shall be used in place of bit 33 of the Real Memory Address as described in that subparagraph, item a.

A serialization function shall be performed before this instruction begins execution. That is, execution of this instruction shall be delayed until all previous central memory accesses on the part of the interrupting processor are complete.

In the event that a processor sends an interrupt to itself, this instruction must complete before the interrupt is taken.

When this instruction attempts execution from a segment not having global privileges, a Privileged Instruction Fault shall be detected, execution of that instruction shall be inhibited, and the corresponding program interruption shall occur.

2.6.4 Monitor Mode

This class of instructions shall be permitted to execute only when the processor is in monitor mode. If an instruction in the monitor mode class attempts execution when the processor is not in monitor mode, an Instruction Specification error shall be detected. Execution of that instruction shall be inhibited, and the corresponding program interruption shall occur.

Monitor mode shall exist whenever the processor is in the state defined by the Monitor Exchange Package. The address contained in the Monitor Process State Register shall point to the Monitor Exchange Package.

Note: No single operation code shall be confined to Monitor mode execution. However, suboperation codes for the instructions defined in 2.6.5 are confined to Monitor mode according to the descriptions contained within that paragraph of this specification.

2.6.5 Mixed Mode

This class of instructions shall include those instructions whose mode is dependent on a parameter selection within the instruction. Depending on the value of the parameter, the mode of the instruction shall be nonprivileged, local privileged, global privileged, or monitor. The description of each instruction shall define which parameter selects the mode and how the selection is made.

2.6.5.1 Branch on Condition Register

Branch to P displaced by 2*Q and alter condition register per jk

9FjkQ (Ref. 134)

This instruction shall test the value of a selected bit in the Condition Register. The j field is used as an index (j of 0 selects bit 48, j of 1 selects bit 49 etc.) into the Monitor Condition Register or within the User Condition Register depending on the k field. The k field shall also determine the branch decision and Condition Register bit alteration as follows:

k = 0 or 8, if bit j of the Monitor Condition Register is set, clear it and take a branch exit.

k = 1 or 9, if bit j of the Monitor Condition Register is not set, set it and take a branch exit.

k = 2 or A, if bit j of the Monitor Condition Register is set, take a branch exit.

k = 3 or B, if bit j of the Monitor Condition Register is not set, take a branch exit.

k = 4 or C, if bit j of the User Condition Register is set, clear it and take a branch exit.

k = 5 or D, if bit j of the User Condition Register is not set, set it and take a branch exit.

k = 6 or E, if bit j of the User Condition Register is set, take a branch exit.

k = 7 or F, if bit j of the User Condition Register is not set, take a branch exit.

Normal Exit - When the test of bit j does not satisfy the branch condition as specified by the k field of this instruction, a normal exit from this instruction shall be performed. A normal exit from this 32-bit instruction shall consist of adding 4 to the rightmost 32 bits of the PVA contained in the P Register, with the sum returned to the P Register's rightmost 32 bits.

Branch Exit - When the test of bit j satisfies the branch condition as specified by the k field of this instruction, a branch exit from this instruction shall be performed. A branch exit shall consist of expanding the Q field from the instruction to 31 bits by means of sign extension, shifting these 31 bits left one bit position with a zero inserted on the right, and adding the 32-bit result to the rightmost 32 bits of the PVA contained in the P Register with the sum returned to the P Register's rightmost 32 bits.

Monitor and Unprivileged Modes - Some values of the k field of this instruction shall cause this instruction to be a Monitor or Unprivileged instruction as follows:

k	Mode
0 or 8	Monitor
1 or 9	Monitor
2 or A	Unprivileged
3 or B	Unprivileged
4 or C	Unprivileged
5 or D	Unprivileged
6 or E	Unprivileged
7 or F	Unprivileged

Unless the processor is in monitor mode when execution is restricted to monitor mode, an Instruction Specification Error shall be detected, execution of this instruction shall be inhibited, and the corresponding program interruption shall occur.

When execution of this instruction results in the setting of a bit in either the monitor condition register or the user condition register, and the corresponding mask bit is set in either the monitor mask register or the user mask register, execution of the instruction shall complete and program interruption shall occur as described in paragraphs 2.8.1 and 2.8.3 of this specification. The PVA stored in the exchange package or stack frame save area by the program interruption shall be the PVA formed from the branch address of the instruction.

2.6.5.2 Copy

These instructions shall provide the means for copying certain state registers to and from X Registers. The state register shall be addressed by means of the rightmost 8 bits initially contained in Register Xj Right.

All state registers and maintenance registers are numbered in table 2.6-1, even though certain registers may be accessed only by the Maintenance Processor, via the Maintenance Access, as specified in table 2.6-2.

Unless the processor is in Monitor Mode when execution is restricted to Monitor Mode, an Instruction Specification Error shall be detected, execution of this instruction shall be inhibited, and the corresponding program interruption shall occur.

Unless the processor is in the appropriately privileged-mode when execution is restricted to local or global privileged mode, a Privileged Instruction Fault shall be detected, the execution of this instruction shall be inhibited, and the corresponding program interruption shall occur.

In the absence of an Instruction Specification Error or Privileged Instruction Fault, the following shall be true:

1. When a "copy" instruction is used to read a nonexistent register or any register which is restricted to Maintenance Processor access only, Register Xk shall be cleared in all 64 bit positions.
2. When a "copy" instruction is used to write a nonexistent register or any register which is "read-only" or restricted to Maintenance Processor access only, such instructions shall result in no operation.

Some implementations of this GDS may not use separate flip-flop registers. Some state registers may be held in central memory even when they are in active use. For such cases, these copy instructions shall make state registers held in central memory appear to operate as copy instructions and not as load or store instructions.

Note: Multiple Address assignments have been specified for certain registers so that, by properly choosing the appropriate address, the contents of a single X Register may be used as both the address and data value for the purpose of copying into such Registers.

Any register less than 64 bits in length shall be copied to or from an X Register as right-justified and zero-filled with respect to the X Register.

Reg. No.*	Register Name	Reference†
00	Status Summary	Model-Dependent
10	Element ID	2.5.1.6
11	Processor ID	2.5.1.8
12	Options Installed	Model-Dependent
13	Virtual Machine Capability List	2.5.1.9
22	Performance Monitoring Facility	2.11
30	Dependent Environment Control	Model-Dependent
31	Control Store Address	Model-Dependent
32	Control Store Breakpoint	Model-Dependent
40	P Register	2.5.2.1
41	Monitor Process State Pointer	2.5.1.2
42	Monitor Condition Register	2.5.2.9
43	User Condition Register	2.5.2.8
44	Untranslatable Pointer	2.5.2.17
45	Segment Table Length	2.5.2.16
46	Segment Table Address	2.5.2.18
47	Base Constant	2.5.2.14
48	Page Table Address	2.5.1.3
49	Page Table Length	2.5.1.4
4A	Page Size Mask	2.5.1.5
50	Model-Dependent Flags	2.5.2.15
51	Model-Dependent Word	2.5.2.25
60	Monitor Mask Register	2.5.2.7
61	Job Process State Pointer	2.5.1.1
62	System Interval Timer	2.5.1.7
63	Keypoint Buffer Pointer	2.5.1.10
80-8F	Processor Fault Status	Model-Dependent
90-9F	Processor Corrected Error Logs	Model-Dependent
A0	Processor Test Mode	Model-Dependent
C0-C3	Trap Enables	2.5.2.20
C4	Trap Pointer	2.5.2.21
C5	Debug List Pointer	2.5.2.23
C6	Keypoint Mask	2.5.2.11
C9	Process Interval Timer	2.5.2.13
CA-CB	Keypoint Enable Flag	2.5.2.5c
E0-E1	Critical Frame Flag	2.5.2.5a
E2-E3	On Condition Flag	2.5.2.5b
E4	Debug Index	2.5.2.22
E5	Debug Mask Register	2.5.2.10
E6	User Mask Register	2.5.2.6

* Register assignment is indicated for model-dependent implementation. If the function exists in the system, the assignment specified shall be used. If the register is not implemented or assignment cannot be adhered to in model-dependent design, the register number shall be reserved.

† Model-dependent registers will be defined in the appropriate model-dependent engineering specifications.

Table 2.6-1. Register Definitions for "Copy"

See section 6 for a further description of the Maintenance Access.

a. **Copy to Xk from state register per Xj**

0Ejk (Ref. 130)

This instruction shall copy the contents of the state register addressed by the contents of Register Xj into Register Xk. The address assignments are defined in table 2.6-1 and the restrictions in table 2.6-2. The Xk Register shall be cleared before the state register is copied into it.

This instruction shall be an <u>unprivileged</u> instruction.

b. Copy to state register from Xk per Xj

0Fjk (Ref. 131)

This instruction shall copy the contents of Register Xk into the state register addressed by the contents of Register Xj. The address assignments are defined in table 2.6-1 and the restrictions in table 2.6-2.

REGISTER NUMBER	PROC-INDEP *	PROC-DEP †	COPY INST. ACCESS PRIVILEGES: COPY FM STATE REGISTER(130) READ REGISTER	COPY INST. ACCESS PRIVILEGES: COPY TO STATE REGISTER(131) WRITE REGISTER	MAINT ACCESS ‡	
00 - 0F	X		no access	no access	R	System Element Independent
10 - 1F	X		unprivileged	no access	R	
20 - 2F	X		no access	no access	R/W	System Element Dependent
30 - 3F		X	no access	no access	R/W	
40 - 4F	X		unprivileged	no access	R/W	
50 - 5F		X	unprivileged	no access	R/W	
60 - 6F	X		unprivileged	Monitor	R/W	
70 - 7F		X	unprivileged	Monitor	R/W	
80 - 8F	X		unprivileged	Global	R/W	
90 - 9F		X	unprivileged	Global	R/W	
A0 - AF	X		unprivileged	Global	R/W	
B0 - BF		X	unprivileged	Global	R/W	
C0 - CF	X		unprivileged	Local	R/W	
D0 - DF		X	unprivileged	Local	R/W	
E0 - EF	X		unprivileged	unprivileged	R/W	
F0 - FF		X	unprivileged	unprivileged	R/W	

* Mandatory implementation, but formats may be model dependent
† Optional implementation
‡ See 6.2.3 for restrictions on Maintenance Processor write

Table 2.6-2. Register Access Privilege

2.6.5.3 Purge

Purge buffer k of entry per Xj

05jk (Ref. 138)

Operation: The Purge Buffer instruction shall invalidate entries in the Map and Cache buffers. The purge may invalidate all entries in a buffer, invalidate all entries in a buffer which derive from a given segment, invalidate all entries in a buffer for a given page, or invalidate all entries in a buffer for a given 512 byte block. Register Xj shall contain the required address information, either System Virtual Address (SVA) or Process Virtual Address (PVA).

An SVA shall contain the Active Segment (ASID) in bits 16 through 31 of Register Xj (bits 0-15 are ignored). A PVA shall contain the Segment number (SEG) in bits 20 through 31 of Register Xj (bits 0-19 are ignored). Bits 32 through 63 shall contain the Byte Number (BN) for either an SVA or a PVA. The rightmost 9 bits of the BN shall be ignored and assumed to be zeros since the smallest purgeable portion of a buffer shall be a 512 byte page or a 512 byte block of a larger page. Proportionately more rightmost bits of the BN shall be ignored and assumed to be zero as page size becomes larger than the 512 byte minimum (k = 8 and A only).

16	20	32	55 63
/////////	SEG	BN	/////////////
ASID		BN	/////////////

The value of k shall determine the buffer to be purged, the range of entries to be purged, and the type of addressing used to determine the range of entries to be purged. The definition of k follows:

k=0 Purge all entries in Cache which are included in the 512 byte block defined by the SVA in Xj.

k=1 Purge all entries in Cache which are included in the ASID defined by the SVA in Xj.

k=2 Purge all entries in Cache. (Contents of Xj are ignored.)

k=3 Purge all entries in Cache which are included in the 512 byte block defined by the PVA in Xj.

k=4 This purge sub-op is in support of the requirement in 2.9.3. As such, the processor shall take one of the following two actions:

- a) Purge all entries in the Instruction Stack (contents of Xj are ignored).
- b) No-op. This is allowable only when the processor purges as per 2.9.3 on the Intersegment Branch.

Certain processors as noted in 1.6 will execute this sub-op as if k=7.

k=5-6 No operation. Certain processors as noted in 1.6 will execute this sub-op as if k=7.

k=7 Purge all entries in Cache which are included in the SEG defined by the PVA in Xj.

k=8 Purge all information from the MAP* pertaining to the one PTE defined by the SVA in Xj.
The size of the page involved shall be determined by the contents of the Page Size Mask Register.

k=9 Purge all information from the MAP* pertaining to the PTEs which are included in the segment defined by the SVA in Xj.

k=A Purge all information from the MAP* pertaining to the PTE defined by the PVA in Xj.
The size of the page involved shall be determined by the contents of the Page Size Mask Register.

k=B Purge all information from the MAP† pertaining to the SDE defined by the PVA in Xj, and to all PTEs included within that segment.

k=C-E No Operation. Certain processors as noted in 1.6 will execute this sub-op as if k=F.

k=F Purge all entries in MAP†. (Contents of Xj are ignored.)

---

* MAP = Page Table MAP on implementations where Page and Segment descriptor entries are kept in separate MAPs.

† MAP = Both Page Table MAP and Segment MAP.

For k=0, 1, 2, 8-F, this instruction shall be a local privileged instruction. It shall be nonprivileged for all other values of k. When this instruction with k=0, 1, 2, or 8-F attempts execution from a segment having neither local nor global privileges, a Privileged Instruction Fault shall be detected, execution of this instruction shall be inhibited, and the corresponding program interruption shall occur.

When k=0-7, the execution of this instruction shall not cause an entry to be made into the cache. However, the fetch of this instruction may cause an entry to be made into cache and the address in Xj may cause entries into the MAP.

When k=8-F, the execution of this instruction shall not cause any entries to be made into the MAP or cache other than any entry associated with the SDE entry associated with the PVA when k=A. However, the fetch of this instruction may cause an entry to be made into the MAP or cache.

Fetching and execution of subsequent instruction along with any loads into MAPs shall be delayed until all central memory accesses and purge operations due to this instruction are completed.

A serialization function shall be performed before this instruction begins execution and again when it completes execution. Execution of this instruction shall be delayed until all previous accesses to central memory, on the part of this processor, are completed. Fetching or execution of subsequent instructions shall be delayed until all central memory accesses and purge operations due to this instruction are completed.

The implementation of this instruction shall be processor model-dependent in that some processor models may not have a Map and/or Cache buffer and they may invalidate more than the required buffer entries. A processor which does not have a Cache shall execute this instruction as a no-operation instruction when cache purges are called for by this instruction. Likewise, a processor which does not have a Map shall execute this instruction as a no-operation instruction when map purges are called for by this instruction. This no-operation for processors without cache or map shall include tests for local privilege, but shall exclude the tests on the PVA and SVA as outlined below. The processor model-dependent specifications shall fully define these model-dependent characteristics.

Note: With respect to the PVA contained in register Xj, Access Violation and Page Table Search without Find conditions described in 2.8.1.7 and 2.8.1.10, shall be excluded and Address Specification Error (bit 32=1) and Invalid Segment conditions, described in 2.8.1.5 and 2.8.1.13 shall be included in the execution of this instruction insofar as Virtual Memory mechanism exception sensing is concerned.

With respect to the SVA contained in register Xj, Access Violation, Page Table Search without Find and Invalid Segment, described in 2.8.1.7, 2.8.1.10 and 2.8.1.13 shall be excluded and Address Specification Error (bit 32=1), described in 2.8.1.5 shall be included in the execution of this instruction insofar as Virtual Memory mechanism exception sensing is concerned.

2.7 PROGRAM MONITORING

2.7.1 Keypoint

Performance of the overall software/hardware system may be monitored via the insertion of Keypoint instructions at "key" points in the software. Each Keypoint instruction shall be identified by its class and by its code within each class. Keypoint classes and keypoint codes may be assigned such that system performance data can be determined from the order and frequency of the occurrence of keypoint instructions of various classes and codes. (See paragraph 2.6.1.7 of this specification.)

2.7.2 Debug

The Debug feature shall allow the testing of the instruction fetches for C180 instructions and the memory references initiated by C180 instructions for virtual memory references which lie within a previously specified set of address ranges. When Debug is enabled, a Trap interrupt will be performed whenever a virtual memory reference matches one of the previously specified set of address ranges. A list of up to 32 address ranges termed Debug List entries may be provided as described in paragraph 2.7.2.1. The specific address tests performed are described in paragraph 2.7.2.2 and appendix G. The scanning of this list for each C180 instruction is described in paragraph 2.7.2.3. Note that Debug testing is performed in C180 state (VMID=0) only and the hardware shall perform no Debug scanning for other virtual machine states.

The term "undefined" in the next paragraphs on debug shall mean undefined only in respect to the debug operation. Thus, when the debug operation is undefined, potential matches may be missed or detected more than once but the execution of the code being debugged shall not be affected as to its integrity.

2.7.2.1 Debug List

The format of a Debug List entry is:

00	08	20	32 63
DC	//////////////	SEG	BN (LOW)
//////////////	//////////////	//////////////	BN (HIGH)
8	12	12	32

where DC is the Debug Code, BN (LOW) is the byte number of the beginning of the contiguous field in memory to which the Debug Code applies, BN (HIGH) is the byte number of the last byte in the contiguous field in memory to which the Debug Code applies, and SEG is the process segment number to which the Debug Code applies. The results of the Compare shall be undefined whenever bit 32 of BN (LOW) or bit 32 of BN (HIGH) is set.

The first entry in the Debug List shall be at the PVA contained in the Debug List Pointer Register (see 2.5.2.23).

Debug List entries shall be aligned on word boundaries. An Address Specification Error shall be recorded whenever a Debug scan is initiated and the PVA contained in the Debug List Pointer is not equal to 0, modulo 8.

The Debug List shall not be longer than 32 entries (64 words). Entries beyond 32 shall be ignored.

The matching of BN (LOW) and BN (HIGH) shall be against the address of the leftmost byte of a piece of information only; whether it is a word, halfword, byte string, or 16-bit instruction. The matching shall include the end points. That is:

BN (LOW) ≤ Address ≤ BN (HIGH).

If BN (LOW) > BN (HIGH) for any Debug List entry, no matching will occur and the scan will proceed to the next double-word entry (if not greater than the maximum of 32 entries or beyond the End of List).

2.7.2.2 Debug Code (DC)

The DC bit assignments are:

Bit 0: Data Read, first address of string - applies to all central memory accesses that are defined as read accesses for purposes of access protection. (See pages G-1 and G-2.)

Bit 1: Data Write, first address of string - applies to all central memory accesses that are defined as write accesses in the memory protection system. (See pages G-3 and G-4.)

Bit 2: Instruction Fetch

Applies to all central memory accesses that are defined as an execution access in the memory protection system. Note that the instruction fetch from memory will have already occurred. (See page G-4.)

Bit 3: Branching Instruction

Applies to branch and return instructions which, when executed will result in a branch exit to the next instruction. The address bracket shall apply to the address of the instruction branched to. (See page G-5 and section 1.6.)

The Compare and Swap instruction (Op. B4) branch exit is not tested (2.6.1.8).

Bit 4: Call Instruction

Applies to the occurrence of either Call instruction. The address bracket shall apply to the address of the called procedure. For Call Indirect (Op. B5) this is the address contained in the Code Base Pointer; for Call Relative (Op. B0) this is the address contained in the modified P Register. (See page G-5.)

Bit 5: End of List

Denotes that this is the last entry in the Debug List.

More than one bit may be set in a DC entry. The End of List DC (bit 5) shall be interpreted after all other bits in the same DC have been interpreted and acted upon.

2.7.2.3 Debug Operation

Debug is enabled whenever all of the following are true:

- Traps are enabled
- UMR56 is set
- VMID is equal to zero

When Debug is enabled, the Debug List shall be scanned after each instruction fetch, but prior to instruction execution, whenever one or more bits in the Debug Mask Register are applicable to the instruction to be executed. (See 2.5.2.10.) Debug List scan will occur irrespective of field length specifications.

Note that performance degradation may begin as soon as Debug is enabled; i.e., Traps are enabled and bit 56 in the User Mask Register is set.

The Debug List shall be scanned by reading the first word from the Debug List in central memory at the PVA specified by the contents of the Debug List Pointer Register. After the first word of the Debug List has been read, each successive word from the Debug List shall be read by incrementing the 6-bit word-index field contained in the Debug Index Register by one, and referencing the Debug List at the PVA specified by the initial contents of the Debug List Pointer Register, modified in its rightmost 32-bit positions by the addition of the zero-extended, 6-bit word-index from the Debug Index Register.

The Debug Index Register, contained in bit positions 00 through 05 of word 36 in the Exchange Package, shall be formatted as follows:

58   63
6-bit word index

When one or more bits contained in the Debug Mask Register are set and are equal to one or more of the corresponding leftmost 5 bits of the Debug Code contained in the first word of a doubleword entry in the Debug List, and one or more of the appropriate PVAs associated with the instruction's execution are contained within the address range defined by the corresponding doubleword entry from the Debug List, the Debug bit in the User Condition Register shall be set, the execution of the instruction shall be inhibited and a trap interrupt shall occur. Moreover, when the End of List bit in Debug Code is set or the 32nd doubleword entry from Debug List has been scanned, the End of List Seen Flag contained in the Debug Mask Register shall be set and no further entries shall be scanned.

The second word of the doubleword Debug List entry which causes a Debug trap interrupt shall be identifiable by the value of the 6 bits of the Debug Index Register and the PVA contained in the Debug List Pointer Register.

The Debug index and flags shall provide the means for properly initiating, resuming and terminating Debug Scan operations, particularly when an instruction's execution has been inhibited by one or more Trap or Exchange interrupts. The Debug flags are End of List Seen and Debug Scan in Progress, bits 9 and 10 of word 36 in the C180 Exchange Package (2.5.2.10).

Exchange interrupts shall cause the flags and Debug Index Register to be stored into the Exchange Package. This allows, for example, a partially completed scan of a Debug Entry list to be resumed.

The processor shall retain the flags and index register on a Trap interrupt so as to allow proper completion of the Debug scan upon return from the interrupt. (If traps are to be enabled during the processing of a Debug Trap interrupt, care must be taken by the software to not reenable Debug or the integrity of the interrupted Debug scan will be lost.)

The scanning of the debug list prior to instruction execution shall include all instruction results except the following which may occur before the Debug scan is complete:

1. Setting page used bit either explicitly as in Test Page Table (Op. 16) or implicitly as with any instruction.
2. Setting of condition register bits.
3. Rounding up of A0 on CALL instruction (Op. B0 or B5).
4. Storing current environment into Stack Frame Save Area on CALL instructions. Note that the DEBUG trap will also round up A0 and store the environment into the SFSA.

The exception testing and DEBUG scan are not constrained to occur in any given sequence relative to each other. There shall be one, and only one, Debug Trap interrupt for each doubleword DEBUG entry having a match or matches. (Two or more matches within the same entry shall produce only one Trap.) The Traps due to exception testing may occur concurrent with a DEBUG Trap (several bits set in MCR and/or UCR) or separately, either before or after the DEBUG scan.

For the purpose of establishing central memory access validation, each central memory access performed for the purpose of reading a word from the Debug List as a part of a Debug Scan operation, shall be a read type access.

2.7.2.4 Software Interface

2.7.2.4.1 Defined Interactions - Debug Enabled

The following items describe the interactions with the Debug facility that are available when Debug is enabled.

- Debug mask bits (11 through 15 of mask register) may be set or cleared via a Copy to State Register instruction and the new bits will be in effect for the Debug scan on the instruction following the Copy Instruction.
- Any Copy to the Debug Flags or Index must clear both flags and any copy to the Index must clear the index or the following Debug scan will be undefined.
- UMR56 may be cleared or Traps disabled via a Copy to State Register instruction with no scan being performed on the instruction following the Copy instruction.
- A Return instruction which loads a User Mask Register with bit 56 clear *or which enters C170 State* will disable Debug with no scan being performed on the instruction following the Return instruction.
- An Exchange operation initiated either by an instruction or by an interrupt will cause the Debug Index, Mask and Pointer registers to be updated as necessary in the Exchange Package so that Debug operation may continue when this process is reinitiated later.
- A Trap operation shall disable Debug by clearing the Trap Enable flag. The processor shall retain the Debug Index, Mask and Pointer so that a Return operation may reenable Debug later.

2.7.2.4.2 Defined Interactions - Debug Not Enabled

2.7.2.4.2.1 Interactions with Debug

The following paragraphs define the allowed interactions with Debug for each of the two types of situations when Debug is not enabled. Any other interaction causes the Debug scan to be undefined for the first instruction encountered after Debug is reenabled.

a. Debug Match Present

The first of these is when Debug is disabled via a Trap of Exchange interrupt and a Debug match occurred at the point of the interrupt. This match will have been the cause or one of the causes of the Trap interrupt or will have been present when a higher priority item caused an Exchange interrupt to occur. The Scan in Progress Flag will be set, the End of List Flag may be set and the Debug Index will indicate the second word of the Debug List entry which produced the match. The End of List Seen Flag being set indicates that the entry which produced the Debug interrupt has the End of List flag set. For this case the following interactions with Debug are defined. (Note that these are either via a Copy to State Register instruction during the Trap interrupt or by writing into the Exchange Package for the Exchange interrupt.

- Any of the Debug mask bits (11 through 15 of the mask register) may be set or cleared and the new mask bit will be in effect for the first Debug scan when Debug is reenabled for this process.
- The Debug Index may be modified by multiples of 2 as long as the final value is greater or equal to 1 and less than or equal to 61. (The Debug Index may only be reset to 0 as described in the next item.)
- The Debug Flags and Index may be cleared to reinitiate the Full Debug scan when Debug is reenabled.
- The End of List Seen Flag may be set to terminate the current Debug scan when Debug is reenabled. The Scan in Progress Flag may, but need not, be altered when setting End of List Seen.
- The End of List Seen Flag may be cleared and Scan in Progress set to continue a scan that had terminated. The Debug Index may also be modified by multiples of 2 as long as the final value is greater or equal to 1 and less than or equal to 61.

b. Debug Match not Present

The other type of situation is when Debug is not enabled for a process and this "not enabled" state arose from other than a Trap or Exchange interrupt with a Debug match present. This situation may arise from any of the following:

- Trap interrupt with no Debug match present.
- Exchange interrupt with no Debug match present.
- Copy to State Register instruction which clears UMR56 or disables Traps.
- CALL to other than C180 environment.
- RETURN to other than C180 environment or which clears UMR56.

For this type of situation, with Debug not enabled, only the following actions are defined. Note that these are either via a Copy to State Register instruction or by writing into the Exchange Package.

- Any of the Debug mask bits (11 through 15 of the mask register) may be set or cleared and the new mask bits will be in effect for the first Debug scan when Debug is enabled for this process.
- The Debug Flags and Index may be cleared, thus initiating a Full Debug scan on the first instruction after Debug is enabled.

2.7.2.4.2.2 Enabling Debug

The Debug operation may be enabled by any one of the four following actions:

- Exchange to a C180 process where traps are enabled and UMR56 is set.
- Return (Op. 04) to a C180 process where the return operation enables traps and UMR56 is set in the user mask register being loaded.
- The software must clear the End of List Seen and Scan in Process flags before executing the return when the following is true:

  a. there are UCR conditions set other than Debug or Process Interval Timer,

     AND

  b. the End of List Seen flag is set,

     AND

  c. the software has altered the PVA for P contained in the Stack Frame Save Area generated by the Trap operation.
- Set UMR56 via Copy to State Register instruction when Traps are enabled. The Debug Flags and Index must be zero prior to execution of the Copy to State instruction or the initial Debug Scan following the Copy to State instruction will be undefined.
- Enable Traps via Copy to State Register instruction when UMR56 is set. The Debug Flags and Index must be zero prior to execution of the Copy to State instruction or the initial Debug Scan following the Copy to State instruction will be undefined.

2.8 PROGRAM INTERRUPTIONS

Numerous conditions may occur that represent program anomalies or other special circumstances so important that the currently executing procedure shall be interrupted and another procedure initiated. As these various interrupt conditions are detected throughout the system, they shall be recorded in or masked by the following four registers:

- Monitor Condition Register (MCR)
- Monitor Mask Register (MM)
- User Condition Register (UCR)
- User Mask Register (UM)

} 16 bits each

An exception is uncorrected errors in systems supporting Stop on Error. In this case, a program interrupt will result in a Stop on Error condition without MCR48 set. Other MCR or UCR bits may be set at the time of the interrupt.

Bits in the Monitor or User Condition Register shall be altered as follows:

- The processor shall set bits to indicate that a particular condition has occurred within the processor or to indicate that the processor has been informed of an event which occurred external to itself.
- Execution of the "Branch and Alter Condition Register" instruction may alter bits.
- A Trap Interrupt Operation will clear any bits for which the associated mask bit is set.
- The Maintenance Processor may alter bits in the MCR or UCR via the Maintenance Access.
- The contents of the MCR or UCR may be altered when held in the Exchange Package in central memory.

Bits in the Monitor User Mask Register may be altered as follows:

- Execution of a "Copy from Xk per (Xj)" instruction which writes into the MM or UM.
- The contents of the MM or UM may be altered when held in the Exchange Package in central memory.
- The Maintenance Processor may alter bits in the MM or UM via the Maintenance Access.

Bits 0 through 6 of the User Mask Register are permanently set and any attempt to clear these bits will be ignored.

The processor shall provide ones for bits 0-6 when the User Mask Register is read or stored into Exchange Packages.

Bit Number and Definition	Associated Monitor Mask Register Bit Set				Mask Bit Clear
	Traps Enabled		Traps Disabled		Traps Enab'd or Disabled
	C180 Job Mode	C180 Mon. Mode	C180 Job Mode	C180 Mon. Mode	C180 Job or Mon. Mode
48 DUE Software Flag	This bit is a flag only and does not cause hardware action for systems with error recovery as specified in 9.1.2.2.				
48 or Detected Uncorrectable Error	Exch	Trap	Exch	Halt	Halt
49 Not assigned	Exch	Trap	Exch	Halt	Halt
50 Short Warning	Exch	Trap	Exch	Stack	Stack
51 Instruction Specification Error	Exch	Trap	Exch	Halt	Halt
52 Address Specification Error	Exch	Trap	Exch	Halt	Halt
53 C170 Exchange Request*	Exch*	Trap*	Exch*	Stack*	Stack*
54 Access Violation	Exch	Trap	Exch	Halt	Halt
55 Environmental Spec. Error	Exch	Trap	Exch	Halt	Halt
56 External Interrupt	Exch	Trap	Exch	Stack	Stack
57 Page Table Search without Find	Exch	Trap	Exch	Halt	Halt
58 System Call	Status - This bit is a flag only & does not cause hdwe. action				
59 System Interval Timer	Exch	Trap	Exch	Stack	Stack
60 Invalid Segment/Ring No. Zero	Exch	Trap	Exch	Halt	Halt
61 Outward Call/Inward Return	Exch	Trap	Exch	Halt	Halt
62 Soft Error	Exch	Trap	Exch	Stack	Stack
63 Trap Exception	Status - This bit is a flag only & does not cause hdwe. action				

* *CYBER 170 State only*

Table 2.8-1. Monitor Condition Register

Tables 2.8-1 and 2.8-2 define the action to be taken when a specific bit is set in one of the two Condition Registers. The action to be taken is a function of the following parameters:

- CY180 <u>Job or Monitor Mode</u> (2.5.1.11)
- <u>Mask Bit Set/Clear</u> - This refers to the state of the mask bit associated with the specific condition bit (2.8.2 and 2.8.4).
- <u>Trap Enabled</u> means

  The Trap Enable flip flop is set "AND"
  The Trap Enable Delay flip flop is clear (see 2.5.2.20).
- <u>Trap Disabled</u> means

  The Trap Enable flip flop is clear "OR"
  The Trap Enable Delay flip flop is set (see 2.5.2.20).

Bit Number and Definition	Associated Monitor Mask Register Bit Set				Mask Bit Clear
	Traps Enabled		Traps Disabled		Traps Enab'd or Disabled
	C180 Job Mode	C180 Mon. Mode	C180 Job Mode	C180 Mon. Mode	C180 Job or Mon. Mode
48 Privileged Instruction Fault	Trap	Trap	Exch	Halt	↑ These mask bits are always set ↓
49 Unimplemented Instruction	Trap	Trap	Exch	Halt	
50 Free Flag	Trap	Trap	Stack	Stack	
51 Process Interval Timer	Trap	Trap	Stack	Stack	
52 Inter-ring Pop	Trap	Trap	Exch	Halt	
53 Critical Frame Flag	Trap	Trap	Exch	Halt	
54 Not assigned	Trap	Trap	Stack	Stack	
55 Divide Fault	Trap	Trap	Stack	Stack	Stack
56 Debug	Trap	Trap	Stack	Stack	Stack
57 Arithmetic Overflow	Trap	Trap	Stack	Stack	Stack
58 Exponent Overflow	Trap	Trap	Stack	Stack	Stack
59 Exponent Underflow	Trap	Trap	Stack	Stack	Stack
60 Fl. Pt. Loss of Significance	Trap	Trap	Stack	Stack	Stack
61 Fl. Pt. Indefinite	Trap	Trap	Stack	Stack	Stack
62 Arithmetic Loss of Significance	Trap	Trap	Stack	Stack	Stack
63 Invalid BDP Data	Trap	Trap	Stack	Stack	Stack

Table 2.8-2. User Condition Register

Tables 2.8-1 and 2.8-2 specify one of the following actions as a result of a specific bit set and the above parameters:

- Exchange (EXCH) - An exchange interrupt to C180 Monitor Mode shall be performed as specified in 2.8.5.1.
- Trap - A trap interrupt shall be performed as specified in 2.8.6.
- Halt - The processor shall stop execution. The Processor Status Summary Register (9.1.2.2) will reflect this condition.
- Stack - The processor shall not interrupt but shall instead continue execution of the current instruction sequence. As the hardware continues execution of additional instructions, this condition may cause a Trap or Exchange interrupt later if the environment changes appropriately.
- Status - This serves to record the occurrence of the specified event but does not directly cause any hardware action to be taken.

A consequence of the interrupt definitions is that for other than instructions having two descriptors no instruction execution will begin until all exceptions arising from the instruction fetch operation have been detected and appropriate action has been taken. This follows naturally from the fact that the instruction itself is not available when exceptions (Invalid Segment, Access Violation, Address Specification Error, Page Table Search Without Find) associated with its fetch are detected. Two

specific instructions which have two descriptors, Move Byte (op code 76) and Compare Byte (op code 77), are allowed but not required to start the fetch of the data described by the first descriptor before the second descriptor has been obtained. This can give rise to exception conditions associated with the fetch of the first data field being reported before an exception (Address Specification Error or Page Fault) associated with the fetch of the second descriptor. Exceptions arising from the execution of the instruction shall not be detected until the second descriptor has been fetched. All of the other two-descriptor instructions shall not report any exceptions associated with the fetch of data until the instruction including both descriptors have been fetched.

There are five types or groups of condition bits as defined in table 2.8-3. This grouping is a function of the characteristics of the event detected. The PVA contained in the P Register at the time the interrupt occurs (and subsequently stored into the Stack Frame Save Area on Trap Interrupts and into the Exchange Package on Exchange Interrupts) is also a characteristic of this grouping. The specific PVA to be stored is described in general below and is specified in detail as part of the description for each Condition bit. (Also see 2.8.7 and 2.8.8.)

Group 0 - This interrupt is a result of an uncorrected hardware malfunction and is called Stop on Error. Stop on Error is the highest priority, unmaskable interrupt. The Maintenance Processor must restart a stopped CPU. Detection of the uncorrected error is a model-dependent function. The PVA in P is undefined except as described in 2.5.2.5. If implemented as specified in 9.1.2.2, MCR48 will not be set.

Stop on Error may be implemented for corrected errors in order to provide more detailed error information. If it is provided, it must be software selectable. The default condition shall be no stop on corrected error.

Certain CPU models listed in 1.6 have implemented Stop on Error along with MCR48. This causes exchanges, traps and halts to occur. Stop on Error is the highest priority interrupt. After being restarted with MCR48 set, the CPU will execute a group 1 interrupt.

Group 1 - This condition results from an uncorrectable hardware malfunction. The detection of this condition (detection = setting bit 48 in MCR and taking appropriate action per table 2.8-1) may occur at any time. The PVA in P is undefined except as described in 2.5.2.5. When Stop on Error is implemented as specified in 9.1.2.2, group 1 interrupts do not exist.

Group 2a* - The hardware generated bits in group 2a (all except Free Flag) occur asynchronously to instruction execution. These shall be detected between instruction executions or trap executions or exchange operations. The PVA in P shall point to the instruction which should have been executed if the interrupt had not occurred. The Free Flag is not set implicitly by hardware.

Group 2b* - These conditions are instruction generated and shall be detected after completion of the instruction. The PVA in P shall point to the instruction which would have been executed if the interrupt had not occurred.

*In the definitions for these events the words: "The PVA contained in P at the time an interrupt occurs shall point to the instruction which would have been executed if the interrupt had not occurred," are intended to mean that on executing an EXCHANGE or RETURN to the interrupted procedure, on completion of the interrupt handling processing will continue at the PVA stored in the exchange package or stack frame save area as if the interrupt had not taken place. For example, if the System Interval Timer decrements to zero during the execution of a logical product instruction, then the execution of that instruction would complete, and then the program would be interrupted. The PVA in P at the time of the interrupt would point at the instruction following the logical product instruction.

Group 3 - These conditions are instruction generated and cause the execution of the instruction to be inhibited. The PVA in P at the time an interrupt occurs points to the instruction which caused the event.

Multiple Group 3 Exceptions -

a. When an instruction contains one or more group 3 MCR exceptions, no group 3 UCR exceptions shall be recorded in the UCR.

b. Multiple Group 3 Exceptions

- Multiple exceptions on one address

  When one address contains more than one of the following four exceptions (Invalid Segment, Access Violation, Address Specification Error, Page Table Search Without Find); the reporting shall be as if the testing is performed in the following order:

```
                          ↓
               ← ← ← ← ← ← ← ← ← ← → →
               ↓                     ↓
        INVALID SEGMENT       ADDRESS SPECIFICATION ERROR
               ↓                     ↓
               ↓                     ↓
        ACCESS VIOLATION             ↓
               ↓                     ↓
               ↓                     ↓
           PAGE TABLE SEARCH WITHOUT FIND
```

  (Thus, for example, and address referencing an Invalid Segment and having an Address Specification Error may report either or both exceptions. An address having an Address Specification Error and a Page Table Search Without Find must report the Address Specification Error and may also report the Page Table Search Without Find, but shall not report the Page Table Search Without Find alone.

  When more than one of these four exceptions are set, only the highest per the following priority order shall be interpreted as valid. All lower priority bits within this group of four are undefined and must be cleared by the software before reinitiating execution of the instruction.

Invalid Segment	(MCR60)
Access Violation	(MCR54)
Address Specification Error	(MCR52)
Page Table Search Without Find	(MCR57)

- Multiple exceptions on multiple addresses within the same instruction

  Instructions having multiple addresses shall follow the requirements above for each individual address. The exceptions for one or all the addresses may be reported via the MCR with the additional requirement that the address associated with the highest priority exception reported (see list above) must be entered into the UTP.

  Thus, the UTP will always contain the address associated with Invalid Segment when MCR60 is one of the multiple exceptions reported. In the absence of MCR60, the UTP will always contain the address associated with Access Violation when MCR54 is one of the multiple exceptions reported. Likewise, Address Specification Error will have precedence over Page Table Search Without Find.

- Multiples involving Instruction Specification Error

  When an instruction contains multiple faults which together give rise to both an Instruction Specification Error and one or more other group 3 MCR exceptions, any one, several, or all of the faults are recorded without preference to the fault causing the Instruction Specification Error. If that particular fault is recorded, however, it is recorded by the Instruction Specification Error, either alone or in any combination with the other exception conditions resulting from that fault. This is also true when an instruction contains a single fault which by itself gives rise to both an Instruction Specification Error and one or more other group 3 MCR exceptions.

- Other

  When multiple group 3 MCR exceptions are present in an instruction and no requirements are listed above; any one, several, or all may be reported (but no group 3 UCR exceptions).

c. Multiple UCR Exceptions

- The BDP instructions and the vector convert floating point to integer are the only instructions which can record more than one group 3 UCR exception other than multiples occurring with debug. (See 2.7.2.3.)

- BDP

  The four exception conditions associated with BDP instructions are listed below:

  - UCR63 Invalid BDP Data
  - UCR57 Arithmetic Overflow
  - UCR62 Arithmetic Loss of Significance
  - UCR55 Divide Fault

  Of these four, the last three are mutually exclusive. Invalid BDP Data and Divide Fault are not recorded together (see 2.3.3). However, Invalid BDP Data may occur with either Arithmetic Overflow or Arithmetic Loss of Significance. In these two cases, Invalid BDP Data must be recorded (in the absence of group 3 MCR exceptions as defined above), and the other exceptions may be recorded.

  When the processor environment is such that (table 2.8-2) no trap will be taken due to the Invalid BDP Data, then the other exception (Arithmetic Overflow or Arithmetic Loss of Significance), if present, must be recorded.

- Vector Convert Floating Point to Integer

  The vector convert instruction (Op. 4C) may encounter FP Indefinite and/or Arithmetic Loss of Significance. This instruction shall record either or both of these exceptions as required by the specific operands.

The detection and subsequent interrupts associated with the exception conditions described above may be illustrated as follows:

Instructions

```
Pre :Inst:Post
Exec:Exec:Exec
     ↓    ↓
     1    2
```

1. Interrupt due to group 3 condition generated by this instruction or due to the detection of a group 2A* condition.
2. Interrupt due to group 2B condition generated by this instruction or due to the detection of a group 2A* condition.

Exchange Operation (including Op. 02)

```
Pre :Exch:Post
Exch:Op'n:Exch
     ↓    ↓
     3    4
```

3. Interrupt due to fetch of the 02 instruction or due to a group 2A* condition.
4. Interrupt due to Environment Specification Error (group 2B) generated by this operation or due to any bit set in the new UCR/MCR or due to the detection of a group 2A* condition.

Trap Operation

```
Pre :Trap:Post
Trap:Op'n:Trap
     ↓    ↓
     5    6
```

5. Interrupts due to group 3 conditions generated by the attempted trap operation or due to the detection of a group 2A* condition.
6. Interrupts due to the detection of a group 2A* condition.

*Note that the above six points are where a group 2A condition <u>may</u> be detected. The requirement for group 2A is that a processor shall detect group 2A conditions between any two instruction executions and/or exchange operations and/or trap operations.

Pre Exec = Pre Execution

That interval of time before the instruction is committed to execution.

Post Exec = Post Execution

That interval of time after instruction execution but before examination of the next instruction.

The following sequence of events could be illustrated as shown below:

A. Exchange instruction in Monitor Mode initiates exchange to Job Mode.

B. Execute two instructions in Job Mode.

C. The second instruction generates a Group 2B condition causing a Trap.

D. The Trap operation encounters a Group 3 condition causing an Exchange.

E. Attempt an instruction in Monitor Mode which generates a Group 3 condition causing a Trap.

```
       A                       B              C         D                E
       ↑                       ↑              ↑         ↑                ↑
 /___________\ /____________________________\ /_______________\ /_____________\
 ┌─────────────┬─────────────┬─────────────┐
 │Pre  :Exch:Post│Pre  :Inst:Post│Pre  :Inst:Post│
 │Exch:Op'n:Exch│Exec:Exec:Exec│Exec:Exec:Exec│
 └─────────────┴─────────────┴─────────────┘
        \______/      \______/
           ↓              ↓           ┌──────── - -
           F              F           │Pre  :Trap
                                      │Trap:Op'n
                                      └──────── - -
                                          ┌─────────────┬──────── - -
                                          │Pre  :Exch:Post│Pre  :
                                          │Exch:Op'n:Exch│Exec:
                                          └─────────────┴──────── - -
                                   \___________/
                                        ↓                  ┌──────────── -
                                        F                  │Pre  :Trap:
                                                           │Trap:Op'n:
                                                           └──────────── -
```

F. Note that asynchronous conditions would be detected at the indicated intervals.

Refer to the following paragraphs for additional detail (2.6.1.2, 2.7.2, 2.8.5, 2.8.6, 2.8.7, and 2.8.8).

GROUP	CONDITION BIT	COND REG. & BIT	TRAP DISABLED MON.MODE	OCCUR C170 STATE	1. Group characteristics 2. When interrupted, the PVA in P points to...
0	Stop on Error	N/A	N/A	Yes	1. Malfunction 2. ...an instruction as defined in 2.8.1.1.
1	Detected Uncorrectable Error	MCR48	Halt	Yes	1. Malfunction 2. ...an instruction as defined in 2.8.1.1.
2A	Short Warning System Interval Timer Soft Error External Interrupt Free Flag Process Interval Timer C170 Exchange Request*	MCR50 MCR59 MCR62 MCR56 UCR50 UCR51 MCR53	Stack Stack Stack Stack Stack Stack Stack	Yes Yes Yes Yes Yes Yes Yes	1. Examined between inst executions because the event is not generated because of inst execution. 2. ...the inst which would have been executed if interrupt had not occurred.
2B (P+)	System Call Exponent Overflow Exponent Underflow FP Loss of Significance Invalid Segment/RN=0 Environment Spec. Error	MCR58 UCR58 UCR59 UCR60 MCR60† MCR55†	N/A Stack Stack Stack Halt Halt	No No No No Yes Yes	1. Examined between instruction executions; the event is generated by the specific instruction. 2. ..the inst following the inst execution during which the specific event occurs.
3 (P)	Instruction Spec. Error Environment Spec. Error Outward Call/Inward Return Trap Exception Invalid Segment/RN=0‡ Access Violation‡ Address Spec. Error‡ Page Table Search wo/Find‡ Privileged Inst'n Fault Unimplemented Instruction Inter-ring Pop Critical Frame Flag Debug FP Indefinite Invalid BDP Data Arith. Loss of Significance Arithmetic Overflow Divide Fault	MCR51 MCR55† MCR61 MCR62 MCR60† MCR54 MCR52 MCR57 UCR48 UCR49 UCR52 UCR53 UCR56 UCR61 UCR63 UCR62 UCR57 UCR55	Halt Halt Halt N/A Halt Halt Halt Halt Halt Halt Halt Halt N/A Stack Stack Stack Stack Stack	No Yes Yes Yes Yes Yes Yes Yes Yes Yes No Yes No No No No No No	1. Pretested before instruction execution; the event is generated by the specific instruction. 2. ...the instruction which caused the event to occur.

**CYBER 170 State only.*

†The following conditions are interpreted as either Group 2B or 3, as noted below:

Reg/Bit	Group 2B	Group 3
MCR60	For Invalid Segment: never For RN=0: Load A, Return, or Pop	For Invalid Segment: all instructions. For RN=0: never
MCR55	Exchange Operations	Call, Return, Pop, & all vector insts.

[See section 1.6 for a list of the C180 models that do not perform the RN=0 test.]

‡See 2.8.7, Multiple Interrupts.

Table 2.8-3. Condition Registers, Bit Grouping

2.8.1 Monitor Condition Register

The Monitor Condition Register (MCR) shall contain 16 bits as defined in table 2.8-1 and the following subparagraphs.

2.8.1.1 Detected Uncorrectable Error (MCR48)

MCR48 is implemented either as DUE Software Flag, which is described in 2.8.1.1.1, or as Detected Uncorrectable Error, which is described in 2.8.1.1.2. The latter is an earlier implementation; CPU models with this implementation are listed in section 1.6. Current and new designs treat MCR48 as DUE Software Flag. See Table 2.8-1.

2.8.1.1.1 MRC48 as DUE Software Flag

MCR48 is a reserved bit for Stop on Error as specified in 9.1.2.2. It is a flag bit that does not cause hardware action. Even though the CPU hardware design ensures that MCR48 is always clear (zero), the software may set this bit as a flag in the central memory exchange package. The PVA contained in P at the time a Stop on Error interrupt occurs is not necessarily the address of the instruction which caused the uncorrected error condition. Proper model-dependent identification of the error condition is required to assess the impact of the uncorrected error. See Process Not Damaged (2.5.2.5.d).

2.8.1.1.2 MCR as Detected Uncorrectable Error

Certain CPU models listed in 1.6 implement MCR48 as follows. The Detected Uncorrectable Error (DUE) bit in the Monitor Condition Register, if set, shall indicate that an uncorrectable error condition has been detected within the processor or on a memory reference initiated by the processor.

These shall include, but not be limited to, the following malfunctions:

- Parity error(s) on transmissions to central memory from this processor.
- Noncorrectable central memory data parity errors (SEC/DED) on central memory accesses from this processor.
- A Bounds Register fault caused by a write operation from this processor.
- Errors detected by an attached ECS Coupler which would not cause a half exit from an ECS instruction. These errors are signaled by the Error End of Operation signal from the ECS coupler (7.13).
- Time-outs as described in paragraph 8.2.9.
- Parity error(s) on transmission from central memory to this processor.
- Other model-dependent conditions as specified in the processor model-dependent specification.

The PVA contained in P at the time a Detected Uncorrectable Error interrupt occurs is not necessarily the address of the instruction which initiated the activity that resulted in the malfunction.

NOTE: If the DUE bit is set and the Process Not Damaged bit (2.5.2.5d) remains clear, other bits in MCR/UCR may set erroneously. As a result of detection of the original uncorrected error, other checkers using the errant data may report errors. For example, a data parity error detected on the input to an arithmetic unit may result in Arithmetic Overflow, setting UCR57.

2.8.1.2 Not Assigned (MCR49)

This bit will not be set implicitly by any hardware condition but may be set/cleared explicitly by software on Exchange or Branch on Condition Register as is the case with any other condition register bit. When set explicitly, this bit causes program interruptions as shown in Table 2.8-1.

2.8.1.3 Short Warning (MCR50)

The Short Warning bit in the Monitor Condition Register shall set as long as a short warning type of environmental failure is present anywhere within the system associated with this processor (see paragraph 9.1.1.2). A processor shall not reflect the short warning condition of other processors in a multiple processor configuration. Only the affected processors shall indicate this condition.

This bit shall clear on any of the clear actions listed in the introductory paragraphs to section 2.8 that occur after the environmental parameter has returned to the normal range.

The PVA contained in P at the time a Short Warning interrupt occurs shall point to the instruction which would have been executed if the interrupt had not occurred.

This bit is forced permanently to zero on some models. See section 1.6.

2.8.1.4 Instruction Specification Error (MCR51)

The Instruction Specification Error bit in the Monitor Condition Register, if set, shall indicate that one of the following errors has occurred.

a. Length Specification errors as described in paragraph 2.2.9 and subparagraphs 2.3.2.1.3 and 2.12.1.2.

b. Type Specification errors as described in paragraph 2.3.3 as well as all type combinations, other than those defined as valid, for the instructions described in each subparagraph of paragraphs 2.3.4 through 2.3.6.

c. Execution of a Program Error instruction as described in subparagraph 2.6.1.1.

d. Execution of a Monitor Mode operation when the processor is not in Monitor mode. (See 2.6.4 and 2.6.5.)

e. Execution of a Call instruction where AT (bit positions 56 through 59 of X0 right) is less than 2. (See 2.6.1.2 and 2.6.1.3.)

The PVA contained in P at the time an Instruction Specification Error interrupt occurs shall point to the instruction with the faulty instruction specification.

2.8.1.5 Address Specification Error (MCR52)

The Address Specification Error bit in the Monitor Condition Register, if set, shall indicate that an attempt was made to use an improper address. Improper addresses shall include:

a. The address modulus defined for specified instructions or specified registers is not met. (See 2.1.3.4, 2.2.1.2, 2.2.1.7, 2.2.3.6, 2.6.1.2 through 2.6.1.5, 2.6.1.8, and 2.12.1.1.)

b. Other address bit(s) defined as zeros for specified instructions or specified registers are not zeros. (See 2.1.5, 2.6.1.10, 2.6.2.1, 2.6.5.3, and 3.2.1.3.)

The PVA contained in P at the time an Address Specification Error interrupt occurs shall point to either one of two places. When the exception is detected during instruction execution, the PVA shall point to the instruction with the faulty address specification. When the exception is detected on an instruction fetch (such as sequential fetches, first fetch following an exchange or branch exit), the PVA shall be the address containing the Address Specification Error. Table 2.1-1 contains a detailed description of the branch operations.

2.8.1.6 CYBER 170 Exchange Request (MCR53)

Systems having a C170 state

The C170 Exchange Request bit in the Monitor Condition Register shall be set once whenever a C170 Exchange Request is received from the IOU. (This bit sets in either C180 or C170 State.) This bit is cleared whenever an Exchange Accept is transmitted to the IOU (or on any of the clear actions listed in the introductory paragraphs to section 2.8).

The C170 Exchange Request initiates one of the C170 Exchange operations as defined in paragraph 7.12.

Bit 53 shall cause the actions as shown in table 2.8-1, but shall not directly cause a C170 Exchange, for example, when set by a Branch on Condition Register instruction. It is a flag to the software that a certain hardware condition has occurred.

The PVA in P at the time the exchange request interrupt occurs shall point to the instruction which would have been executed had the interrupt not occurred.

Systems not Having a C170 state

This bit will not be set implicitly by any hardware condition but may be set/cleared explicitly by software on Exchange or Branch on Condition Register as is the case with any other condition register bit. When set explicitly, this bit causes program interruptions in a manner identical to bit 50 of the MCR.

2.8.1.7 Access Violation (MCR54)

The Access Violation bit in the Monitor Condition Register, if set, shall indicate that the requested memory access was blocked because it did not have the required access permission. See section 3.6 of this specification for details. Access violations shall be detected for the following central memory access situations.

a. Read central memory when read access is not granted or read is not within read ring limits.
b. Write central memory when write access is not granted or write is not within write ring limits.
c. Attempt to execute when execute access is not granted or execute is not within execute ring limits.
d. Call via a code base pointer which is not in a binding section segment. (See 2.6.1.2.)
e. Call from a process beyond the call ring limit. (See 2.6.1.2.)
f. Key/lock violations. See section 3.6.3.2 for the definition of key/lock violations.

Also note the requirements in paragraphs 2.2.1.4, 2.2.3.6, 2.6.1.10, 2.6.2.1, and 2.6.5.3. The PVA contained at P at the time an Access Violation interrupt occurs shall point to either one of two places. When the exception is detected during instruction execution, the PVA shall point to the instruction which made the central memory access that attempted to violate the access protection mechanism. When the exception is detected on the first instruction following an exchange, the PVA shall be the address associated with the Access Violation.

2.8.1.8 Environment Specification Error (MCR55)

The Environment Specification Error bit in the Monitor Condition Register, if set, shall indicate that an error was detected in the environment as described below.

- The PVA contained in P at the time an environment error interrupt occurs shall point to the instruction which caused the Environment Specification Error when:

  a. A mismatch between VMCL and the VMID obtained from the Code Base Pointer (2.5.5.1) on a CALL instruction (Op. B5).

  b. A mismatch between VMCL and the VMID obtained from the Stack Frame Save Area (2.5.4) on a RETURN instruction (Op. 04).

  c. Initial A2 (Previous Save Area Pointer) not equal to A0 (Dynamic Space Pointer) in the Stack Frame Save Area on a RETURN (Op. 04) or POP (Op. 06) instruction.

  d. The value of the field designating the last A register to be loaded, as contained in the Previous Stack Frame Descriptor, is less than 2 on a RETURN instruction (Op. 04).

  e. Execution of a vector instruction was attempted with a page size less than 4096 bytes or with RMA bit 33 set (thus, directing reference to shared memory) on a processor with the vector option installed.

- The PVA contained in P at the time an environment error interrupt occurs shall point to the instruction which would have been executed had the interrupt not occurred when:

  a. A mismatch between VMCL and the VMID obtained from the Exchange Package on a Monitor to Job or Job to Monitor. (P points to the first instruction that would have been executed following the Exchange.)

  b. An attempt to execute a Return instruction with some data in the SFSA Pushdown but the SFSA is not accessible in central memory. In this case, the processor may but need not also report the exception condition which was encountered. If an Address Specification Error, Page Table Search Without Find, Invalid Segment, or Access Violation is reported along with the Environment Specification Error, the UTP must be appropriately loaded. Detection of an Environment Specification Error because the SFSA is not accessible may cause the hardware to leave garbage in the A and X registers and in the TOS pointer for the target ring. The hardware (including the Service Processor) shall leave the P of the process to which control was being passed in the P register and shall not leave an A register with a ring number less than the P ring number. The environment is damaged and the process cannot be restarted.

- The PVA contained in P at the time an Environment Specification Error interrupt occurs shall point to the instruction as defined under the condition causing the TRAP operation when:

  a. A mismatch between VMCL and the VMID obtained from the Code Base Pointer (2.5.5.1) on a TRAP operation (2.8.6).

  b. External Procedure Flag not set in the Code Base Pointer when executing a TRAP operation.

  c. The VMID from the Code Base Pointer not equal to zero when executing a TRAP operation (2.8.6).

2.8.1.9 External Interrupt (MCR56)

The External Interrupt bit in the Monitor Condition Register, if set, shall indicate the receipt of an interrupt from a processor. (The recipient processor may read a message in central memory to determine who the calling processor is and the purpose of the external interrupt.)

The PVA contained in P at the time an External Interrupt occurs shall point to the instruction which would have been executed if the interrupt had not occurred.

2.8.1.10 Page Table Search Without Find (MCR57)

The Page Table Search Without Find bit in the Monitor Condition Register, if set, shall indicate that the requested page table entry was not found during the linear search of the page table which begins at the "hashed" entry address and ends a maximum of 32 entries later. Thus, the system virtual address could not be mapped into a real memory address. (See 2.1.3.4, 2.2.3.6, 2.6.1.10, 2.6.2.1, 2.6.5.3 and 3.5.2.)

The PVA associated with P for a Page Table Search Without Find interrupt shall point to the instruction which attempted the central memory access which resulted in the Page Table Search Without Find condition. (See table 2.1-1.)

The PVA associated with P for a Page Table Search Without Find interrupt shall point to the instruction which attempted the central memory access which resulted in the Page Table Search Without Find condition when the exception is detected during instruction execution, or shall be the address associated with the Page Table Search Without Find when the exception is detected on an instruction fetch (such as sequential fetches, first fetch following an exchange or branch exit). Table 2.1-1 contains a detailed description of the branch operations.

2.8.1.11 System Call (MCR58)

The System Call bit in the Monitor Condition Register, if set, shall indicate that a C180 process has executed an Exchange instruction (Op. 02) which caused an exchange interrupt from job to monitor *or that a C170 process has executed a Central Exchange Jump (Op. 013) which was followed by an exchange interrupt to C180 monitor due to the C170 X0 sign bit being set. (See 7.3.7.)* This bit shall not be set by an Exchange instruction (Op. 02) going from monitor process state to job process state. (See 2.6.1.6.)

The PVA associated with P for a System Call interrupt shall point to the instruction which would have been executed if the interrupt had not occurred; i.e., the PVA of the instruction immediately following the Exchange instruction.

2.8.1.12 System Interval Timer (MCR59)

The System Interval Timer bit in the Monitor Condition Register, if set, shall indicate that the System Interval Timer has decremented to a count equal to zero. (See 2.5.3.2.)

The PVA contained in P at the time a System Interval Timer interrupt occurs shall point to the instruction which would have been executed if the interrupt had not occurred.

2.8.1.13 Invalid Segment/Ring Number Zero (MCR60)

The Invalid Segment/Ring Number Zero bit in the Monitor Condition Register, if set, shall indicate that an error was detected as described below.

Invalid Segment

The Invalid Segment bit in the Monitor Condition Register, if set, shall indicate that a PVA could not be translated into a Real Memory Address because the Segment Table Length was exceeded or because the Segment Descriptor was invalid (3.3). Exceptions are noted in 2.6.2.1 and 2.6.5.3.

The PVA contained in P at the time an Invalid Segment interrupt occurs shall point to the instruction which attempted to reference an Invalid Segment when the exception is detected during instruction execution or shall be the address associated with the Invalid Segment when the exception is detected on the first instruction fetch following exchange.

Ring Number Zero

The Ring Number Zero bit in the Monitor Condition Register, if set, shall indicate that an A register was loaded with a PVA whose RN=0 via a Return (2.6.1.4), a Pop (2.6.1.5), a Load A (2.2.1.6), a Load A Indexed (2.2.1.6), or a Load Multiple (2.2.1.7).

The PVA contained in P at the time a Ring Number Zero interrupt occurs shall point to the instruction which would have been executed if the interrupt had not occurred. [See section 1.6 for a list of the C180 models that do not perform the RN=0 test and will only set MCR in the event of Invalid Segment.]

2.8.1.14 Outward Call/Inward Return (MCR61)

The Outward Call/Inward Return bit in the Monitor Condition Register, if set, shall indicate that an outward call or an inward return has been attempted by the processor. An outward call shall have been attempted if the Call instruction attempts a call to a procedure with a ring number larger than the ring number of the procedure which contains the call instruction. An inward return shall have been attempted if the return instruction attempts a return to a procedure with a ring number smaller than the ring number of the procedure which contains the return instruction. (See 2.6.1.2 and 2.6.1.4.)

The PVA in P when an Outward Call/Inward Return interrupt occurs shall point to the instruction which attempted the outward call or inward return.

2.8.1.15 Soft (or Corrected) Error (MCR62)

The Soft Error bit in the Monitor Condition Register, if set, shall indicate that the hardware has detected and corrected a hardware malfunction as described below:

a. A reference to central memory from this processor results in either a WRITE CORRECTED ERROR or a READ CORRECTED ERROR response from central memory (4.2.2). Specific information about the corrected error is contained in the central memory Corrected Error Log (4.5.1.6).

b. A Corrected Error signal is received from the ECS Coupler during the execution of a CY170 ECS instruction (7.2.3, 7.13.3).

c. The hardware detection and correction of an error caused by a hardware malfunction within the processor shall also set the Soft Error bit. The detection of an error in logic unused by the current instruction may also give rise to a soft error. This includes the successful retry of instructions. See the appropriate processor model-dependent specification for details.

The PVA contained in P at the time a Soft Error interrupt occurs shall point to the instruction which would have been executed if the interrupt had not occurred.

2.8.1.16 Trap Exception (MCR63)

The Trap Exception bit in the Monitor Condition Register, if set, shall indicate that a fault was detected during the trap interrupt operation. The fault detected shall be indicated by setting the appropriate bit in the Monitor Condition Register. (See 2.8.6.)

The PVA contained in P at the time a Trap Exception interrupt occurs shall be the PVA which would have been stored in the stack frame save area, word zero, had the trap completed without any exceptions.

2.8.2 Monitor Mask Register

The Monitor Mask Register (MM) shall contain 16 bits, each of which is the mask bit for its respective bit (48-63) of the MCR.

2.8.3 User Condition Register

The User Condition Register (UCR) shall contain 16 bits as defined in table 2.8-2 and the following subparagraphs.

2.8.3.1 Privileged Instruction Fault (UCR48)

The Privileged Instruction Fault bit in the User Condition Register, if set, shall indicate that one of the following faults has occurred.

a. An attempt was made to execute a local privileged instruction in other than local privileged executable mode or in global privileged executable mode. (See 2.6.2.)

b. An attempt was made to execute a global privileged instruction in other than global privileged executable mode. (See 2.6.3.)

c. *An attempt was made to execute a CYBER 170 017 instruction in the C170 state machine. (See 7.3.2.)*

The PVA associated with P shall point to the instruction which caused the Privileged Instruction Fault interrupt to occur.

2.8.3.2 Unimplemented Instruction (UCR49)

The unimplemented instruction bit in the User Condition Register, if set, shall indicate that an instruction operation code which is not implemented in a particular processor model has attempted execution in that processor model. The implementation of this bit is processor model-dependent and shall be fully specified in the appropriate processor model-dependent specifications.

Attempted execution of C170 Op. Codes 464, 465, 466 and 467 (the Compare/Move instructions described in paragraph 7.3.1) when in the C170 state machine (VMID=1) shall cause this bit to be set on certain models as specified in section 1.6.

The PVA associated with P shall point to the Unimplemented Instruction which caused the interrupt to occur.

2.8.3.3 Free Flag (UCR50)

The Free Flag bit in the User Condition Register, if set, shall indicate that this process shall take immediate note of a situation which occurred when this process was not in active execution.

A process' free flag shall normally be set in the process' exchange package when the exchange package is in central memory. In this way, a system process shall gain the object process' immediate attention the next time the object process begins active execution.

The PVA associated with P shall point to the instruction which would have been executed if the Free Flag interrupt had not occurred.

2.8.3.4 Process Interval Timer (UCR51)

The Process Interval Timer bit in the User Condition Register, if set, shall indicate that the Process Interval Timer has decremented to zero. (See 2.5.3.1.)

The PVA associated with P shall point to the instruction which would have been executed if the Process Interval Timer interrupt had not occurred.

2.8.3.5 Inter-ring Pop (UCR52)

The Inter-ring Pop bit in the User Condition Register, if set, shall indicate that an attempt was made to "pop" a stack frame in one ring with a Pop instruction (reference number 118) executing in a different ring. (See 2.6.1.5.)

The PVA associated with P shall point to the Pop instruction which attempted the inter-ring pop.

2.8.3.6 Critical Frame Flag (UCR53)

The Critical Frame Flag bit in the User Condition Register, if set, shall indicate that an attempt was made to "pop," or "return" from, a critical stack frame. (See 2.5.2.5, 2.6.1.4, 2.6.1.5, and 2.8.10.)

The PVA associated with P shall point to the Return instruction or the Pop instruction which attempted to "pop," or "return" from, a critical stack frame.

2.8.3.7 Not Assigned (UCR54)

This bit will not be set implicitly by any hardware condition but may be set/cleared explicitly by software on Exchange or Branch on Condition Register as is the case with any other condition register bit. When set explicitly, this bit causes program interruptions as shown in Table 2.8-2.

The PVA associated with P shall point to the next instruction which would have been executed if the interrupt had not occurred.

2.8.3.8 Divide Fault (UCR55)

For the definition of this condition, see the instruction descriptions in subparagraphs 2.2.2.4, 2.2.2.8, 2.3.3, 2.4.1.9.3 and 2.4.1.9.6

When Divide Fault occurs during the execution of a nonvector instruction, the PVA associated with P shall point to the instruction which caused the divide Fault to occur.

When Divide Fault occurs during the execution of a vector instruction, the PVA associated with P shall point to the instruction which caused the Divide Fault condition to occur, unless there is another interrupt condition which dictates that the PVA associated with P shall point to the instruction following the one which yielded the divide fault. In this case, the PVA associated with P shall point to the instruction following the one which yielded the divide fault.

2.8.3.9 Debug (UCR56)

For the definition of this condition, see the debug description in section 2.7.2. The debug operation shall not set this bit unless Traps are enabled and the mask bit is set.

The PVA associated with P shall point to the instruction which caused the Debug interrupt to occur. (For the purpose of this definition an instruction fetch shall be considered part of the execution of that instruction and a branch taken shall be considered part of the execution of the branch instruction.)

2.8.3.10 Arithmetic Overflow (UCR57)

For the definition of this condition, see the instruction descriptions in subparagraphs 2.2.2.1 through 2.2.2.8, 2.3.3.1, and 2.3.6.3.

When Arithmetic Overflow occurs during the execution of a nonvector instruction, the PVA associated with P shall point to the instruction which caused the Arithmetic Overflow to occur.

When Arithmetic Overflow occurs during the execution of a vector instruction, the PVA associated with P shall point to the instruction which caused the Arithmetic Overflow condition to occur, unless there is another interrupt condition which dictates that the PVA associated with P shall point to the instruction following the one which yielded the arithmetic overflow. In this case, the PVA associated with P shall point to the instruction following the one which yielded the arithmetic overflow.

2.8.3.11 Exponent Overflow (UCR58)

For the definition of this condition, see the descriptions in subparagraphs 2.4.1.9.1 through 2.4.1.9.6.

The PVA associated with P shall point to the instruction which would have been executed if the Exponent Overflow trap interrupt had not occurred, that is, the instruction following the one which yielded the exponent overflow condition.

2.8.3.12 Exponent Underflow (UCR59)

For the definition of this condition, see the descriptions in subparagraphs 2.4.1.9.1 through 2.4.1.9.6.

The PVA associated with P shall point to the instruction which would have been executed if the Exponent Underflow trap interrupt had not occurred, that is, the instruction following the one which yielded the exponent underflow condition.

2.8.3.13 Floating Point Loss of Significance (UCR60)

For the definition of this condition see subparagraphs 2.4.1.9.1 and 2.4.1.9.4.

The PVA associated with P shall point to the instruction which would have been executed if the Floating Point Loss of Significance trap interrupt had not occurred, that is, the instruction following the one which yielded the floating point loss of significance condition.

2.8.3.14 Floating Point Indefinite (UCR61)

For the definition of this condition see subparagraphs 2.4.1.9.1 through 2.4.1.9.6.

When Floating Point Indefinite occurs during the execution of a nonvector instruction, the PVA associated with P shall point to the instruction which caused the Floating Point indefinite to occur. (Also see 2.8.7 and 2.8.8.)

When Floating Point indefinite occurs during the execution of a vector instruction, the PVA associated with P shall point to the instruction which caused the Floating Point Indefinite condition to occur, unless there is another interrupt condition which dictates that the PVA associated with P shall point to the instruction following the one which yielded the Floating Point Indefinite. In this case, the PVA associated with P shall point to instruction following the one which yielded the Floating Point Indefinite.

2.8.3.15 Arithmetic Loss of Significance (UCR62)

For the definition of this condition, see paragraph 2.3.3, and subparagraphs 2.3.3.2, 2.3.3.3, 2.3.6.1, and 2.4.1.8.2.

When Arithmetic Loss of Significance occurs during the execution of a nonvector instruction the PVA associated with P shall point to the instruction which caused the Arithmetic Loss of Significance to occur. (Also see 2.8.7 and 2.8.8.)

When Arithmetic Loss of Significance occurs during the execution of a vector instruction, the PVA associated with P shall point to the instruction which caused the Arithmetic Loss of Significance condition to occur, unless there is another interrupt condition which dictates that the PVA associated with P shall point to the instruction following the one which yielded the Arithmetic Loss of Significance. In this case, the PVA associated with P shall point to the instruction following the one which yielded the Arithmetic Loss of Significance.

2.8.3.16 Invalid BDP Data (UCR63)

For the definition of the condition see paragraph 2.3.3 as well as subparagraphs 2.3.4.5, and 2.3.6.1 through 2.3.6.3.

The PVA associated with P shall point to the instruction which caused the invalid BDP condition. (Also see 2.8.7 and 2.8.8.)

2.8.4 User Mask Register

The User Mask Register (UM) shall contain 16 bits, each of which is the mask bit for its respective bit (48-63) of the UCR. Bits 48 through 54 are permanently set and any attempt to clear these bits shall be ignored.

2.8.5 Exchange Operation and Interrupts

Exchange operations are those in which a processor changes either from the job process state to the monitor process state or vice versa. Exchange interrupts specified in tables 2.8-1 and 2.8-2 shall cause an Exchange operation only from C180 job process state to C180 monitor process state as reflected in tables 2.8-1 and 2.8-2.

The exchange package (see figure 2.5-1) shall be contained in central memory at separate locations for each process. The exchange package shall be used to establish the environment for each process when the process is activated. An exchange operation shall deactivate one process and activate a second process.

The exchange operation shall consist of moving the environment for the current process state into its central memory locations and establishing the environment for the next process state by moving it from its central memory locations. The only exception condition generated by the execution of an Exchange operation is an Environment Specification Error as described in 2.5.6. The Exchange is allowed to complete, the Environment Specification Error is recorded in the Monitor Condition Register, and then the Monitor and User Condition Registers are examined before instruction execution.

Upon completion of an exchange operation, any model-dependent instruction stacks (buffers) shall be cleared. The initial fetching of each instruction following an exchange operation shall be from central memory or from the cache buffer. Cache shall be bypassed when executing processor exchange operations. Information in cache shall be addressed by the System Virtual Address (SVA) and shall not be purged as the result of an exchange operation.

At the completion of the Exchange operation, bits may be set in UCR and/or MCR for only the following reasons:

1. Bits set in UCR/MCR as contained in the Exchange Package as loaded from central memory.
2. Environment Specification Error set because the VMID in the loaded Exchange Package does not match VMCL.
3. Group 1 or 2A bit set due to the occurrence of asynchronous event.

The number of items in the exchange package held in registers when a state is active shall be processor model-dependent and shall be fully specified in the processor model-dependent specifications.

The PVA stored in the P Register portion of the Exchange Package, for each condition that can cause an exchange interrupt, is defined in each Condition Register bit definition (see 2.8.1 and 2.8.3). The same definition for the PVA stored shall apply to a trap interrupt, except that P shall be stored in the Current Stack Frame Save Area, Word 0. (Also see 2.8.7 and 2.8.8.)

2.8.5.1 Job Process to Monitor Process Exchange

The hardware shall perform the following steps when doing a job process to monitor process exchange (see 2.6.1.6).

a. Store the current job process state exchange package in central memory beginning at the address contained in the job process state pointer register.

b. Disable Exchange interrupts.

c. Load the monitor process state exchange package from central memory, beginning at the address contained in the monitor state pointer register, into the processor environment registers. (See 2.5.6 for virtual machine support.)

Exchange interrupt conditions which occur in the monitor process state while traps are enabled shall be trapped.

Exchange interrupt conditions which occur in the monitor process state while traps are disabled shall be held until traps are enabled, in which case, a trap shall be taken. For those cases in which continued processor execution is impossible or likely to destroy information, the processor shall halt and set bit 60 of the Processor Status Summary Register.

An exchange to C180 Monitor Mode will always have one or more bits set in the MCR and/or UCR stored in the exchange package at job process state except for the following two events which need not leave any bits set in MCR and/or UCR:

- *The conversion of a C170 Halt into a C180 Exchange as described in paragraph 7.6.1.*
- A half exchange initiated via MAC as described in 6.1.2.3.

See tables 2.8-1 and 2.8-2 for the definition of how the conditions are handled under various circumstances.

2.8.5.2 Monitor Process to Job Process Exchange

The hardware shall perform the following steps when performing a monitor process to job process exchange (see 2.6.1.6).

a. Store the monitor process state exchange package in central memory beginning at the address contained in the monitor process state pointer register.

b. Enable exchange interrupts.

c. Load the job process state exchange package into the processor environment registers from central memory beginning at the address contained in the job process state pointer register.

Notes: The monitor process shall establish the next job process for execution by loading the job process state pointer register with the central memory location of the next job's exchange package. (See 2.5.6 for virtual machine support.)

The hardware shall not allow any group 2A event to set a bit in the MCR/UCR (whose associated mask bit is set) at such a time that an exchange operation from Monitor Mode (with traps enabled) to Job Mode shall cause the bit in MCR/UCR to be stored as a part of the monitor exchange package. Either the trap must be taken in Monitor Mode or the setting of the MCR/UCR bit due to the asynchronous event shall be deferred to Job Mode.

2.8.6 Trap Interrupt Operation

Trap Interrupts shall be accomplished by simulating a Call Indirect instruction (Op. B5) to an external procedure and shall include virtual machine support as described in paragraph 2.5.6.

A Trap Frame shall be established in the manner described in subparagraph 2.5.4.1 of this specification. This Trap Frame shall be used to store the "environment" of the "trapped" procedure.

Code Base and Binding Section Pointers shall be obtained by using the PVA contained in the Trap Pointer Register in place of the "(Aj) plus 8*Q" as utilized by the explicit Call instruction, (described in subparagraph 2.6.1.2 of this specification), which the Trap Interrupt shall simulate.

If an exception condition arises during the execution of a trap interrupt, that trap shall be aborted and the following actions shall take place:

1. The Trap Exception bit shall be set in the MCR (2.8.1.16).
2. The appropriate MCR/UCR bit shall be set for the exception condition that caused the Trap to abort.
3. The Exchange or Halt (as specified in tables 2.8-1 and 2.8-2 with Traps considered disabled) shall be performed. In the case of an Exchange operation, the Trap Enable flip-flop remains set in the Exchange Package stored for the interrupted procedure.

The UCR/MCR as stored into memory on the Exchange will always contain at least three bits:

- Bit or bits which initiated the Trap operation (MCR and/or UCR)
  AND
- Trap Exception bit (MCR)
  AND
- Bit or bits (MCR) which caused the Trap operation to abort.

This shall include:

- Address Specification Error, or
- Access Violation, or
- Invalid Segment/Ring Number Zero, or
- Page Table Search Without Find, or
- Environment Specification Error, or
- Outward Call.

Any of the asynchronous monitor or system conditions contained in group 1 or 2A may, but need not, cause the Trap operation to abort [MCR48, MCR50, *MCR53 (C170 Exchange Request),* MCR56, MCR59 or MCR62].

If no exception conditions arise, the Trap operation shall disable traps (clear the Trap Enable flip-flop). Traps may be reenabled by software either by setting the Trap Enable flip-flop and the Trap Enable Delay flip-flop and issuing a Return instruction (Op. 04) as described in paragraph 2.5.2.20, or by setting the Trap Enable flip-flop. Note that the Trap Enable flip-flop and the Trap Enable Delay flip-flop may be set simultaneously by a single copy instruction. See paragraph 2.6.5.2 of this specification.

The Call instruction shall be simulated by means of the sequence described in 2.6.1.2 with the following differences:

1. Omit step 1.
2. Step d is accomplished as follows: the rounded Dynamic Space Pointer contained in Register A0, shall be increased by 264 (decimal) and the result shall be stored into the process Exchange Package as the Top of Stack pointer corresponding to the ring of execution of the "trapped" procedure.
3. Unless the Code Base Pointer's External Procedure Flag is equal to a one, an Environment Specification Error shall be detected and an Exchange Interrupt or a processor halt shall occur (see table 2.8-1).
4. Unless the Code Base Pointer's VMID=0, an Environment Specification Error shall be detected and either an Exchange Interrupt or a processor halt shall occur (see table 2.8-1).
5. The Monitor and User Condition registers are stored in the Stack Frame Save Area (see subparagraph 2.5.4.1), after which each bit in these two registers is cleared where the associated mask bit is set.
6. In contrast to the CALL instruction which places data into the SFSA Pushdown on processors having a SFSA Pushdown (see 1.6), the TRAP operation will purge the contents of SFSA Pushdown.

When not executing a TRAP operation, all bits in the Condition Registers which are identified as trap interrupts shall cause a trap interrupt when set, under the circumstances described in tables 2.8-1 and 2.8-2.

User processes shall have control over whether a user condition will cause a trap, via the User Mask Register. Bits in the User Mask Register when set shall permit corresponding User Condition bits to trap. Several of the User Mask Register bits shall be permanently set as specified in paragraph 2.8.4.

Trap Conditions which occur when traps are not enabled shall in some cases result in Exchange interrupts when in Job Mode and Processor halts when in Monitor Mode. Table 2.8-2 defines how each User Condition bit is treated under these circumstances.

The PVA stored in the P Register portion of the Current Stack Frame save area, for each condition that can cause a trap interrupt, is defined in each Condition Register bit definition in paragraphs 2.8.1 and 2.8.3.

2.8.7 Multiple Interrupts

When more than one bit is set in the MCR and/or UCR, the following priority shall be observed regarding the translation of those bits: (see tables 2.8-1 and 2.8-2)

1. HALT. If any of the bits call for a Halt, the processor shall perform a Halt operation regardless of which other bits may or may not be set.
2. EXCHANGE. If no bits call for a Halt, the processor shall perform an Exchange operation if any of the bits call for an Exchange.
3. TRAP. If no bits call for either a Halt or an Exchange, the processor shall perform a Trap operation if any of the bits call for a Trap.
4. STACK. If no bits call for either a Halt or an Exchange or a Trap, the processor shall perform a Stack operation (not an interrupt) if any of the bits call for a Stack.

With reference to the bit groupings shown in table 2.8-3, the PVA as stored into the Exchange Package, or into the Stack Frame Save Area shall be determined as follows:

1. Whenever a group 1 condition occurs, P is undefined. (See 2.8.1.1.)
2. In the absence of group 1 conditions, P shall be as defined for the instruction generated interrupt, that is, for group 2b or 3 rather than for group 2a.

The execution of a scalar instruction cannot directly cause the recording of both group 2b and group 3 conditions because group 3 conditions are detected and the appropriate action is taken before execution, while group 2b conditions are recorded after instruction execution. (In certain circumstances, however, a scalar instruction may cause the detection of a group 2b condition which, in turn, causes an attempted Trap operation that does not complete due to its own group 3 exception condition. This particular sequence will create condition registers containing both group 2b and group 3 conditions, but only the group 2b conditions were generated directly by the instruction itself, and the PVA in P matches the definition for the group 2b conditions.)

The execution of a vector instruction can directly cause the recording of both group 2b and group 3 conditions. For this case, P shall be defined for group 2b. (See 2.12.1.6.)

Instruction generated condition bits (groups 2b and 3) shall be examined on an instruction by instruction basis. That is, no bits shall be set in the MCR and/or UCR by an instruction when there is a required program interruption due to bits set by a previous instruction. If multiple bits in groups 2b or 3 are loaded into MCR and/or UCR during an Exchange operation, the PVA in P shall be as specified in paragraph 2.8.8.

2.8.8 Enabling Interrupts

When a bit is set in a condition register by a "Branch on Condition Register" instruction (see paragraph 2.6.5.1) and the corresponding bit is set in either the Monitor Mask Register or the User Mask Register, then the PVA in P at the time of the interrupt shall point to the instruction branched to by the "Branch on Condition Register" instruction.

When an interrupt condition is stacked because the corresponding bit in either the Monitor Mask Register or the User Mask Register is clear, and the interrupt is subsequently enabled, by setting the appropriate bit in either the Monitor Mask Register or User Mask Register; then an interrupt shall occur, and the PVA in P at the time of the interrupt shall point to the instruction following the instruction which enabled the interrupt.

When an interrupt condition is stacked because the traps are not enabled and then traps are subsequently enabled, then the interrupt shall occur, and the PVA in P at the time of the interrupt shall point to the instruction following the instruction which enabled the traps.

Following an exchange from Monitor to Job state, the Free Flag is examined.

When the Free Flag is set, the processor shall examine the MCR/UCR as loaded from memory, plus only the potential Environment Specification Error from the exchange operation.

When the Free Flag is clear, the processor shall examine the MCR/UCR as loaded from memory plus the potential Environment Specification Error from the exchange operation; optionally, the processor may also examine any group 3 exceptions associated with the fetch and attempted execution of the first instruction. If an Exchange or Trap takes place, the PVA in P at the time of the interrupt shall point to the instruction which would have been executed following the Exchange had that initial interrupt not occurred.

2.8.9 Interrupt Flowchart

The flowchart at the end of this paragraph diagrams the process which detects an exception condition and takes action on it.

The occurrence of an exception is indicated by the presence of a one bit in one of two registers: the monitor condition register or the user condition register; *an exception is the C170 Halt bit as described in 7.6.1.* In practice there are four classes of exception conditions which have been grouped into two registers for software convenience. The four classes are:

(i) **Monitor Conditions**

These are exception conditions, directly related to the active process, which preclude further processing until corrective measures, if possible, are taken. They are:

Detected Uncorrectable Error
Instruction Specification Error
Address Specification Error
Access Violation
Environment Specification Error
Page Table Search Without Find
Invalid Segment
Outward Call/Inward Return
Unimplemented Instruction
Privileged Instruction Fault
Inter-ring Pop
Critical Frame Flag

The last four of these conditions are flagged in the user condition register to permit the user to receive control in a user supplied trap routine.

(ii) **System Conditions**

These are exception conditions not directly related to the active process, which do not preclude further processing. They are:

Short Warning
External Interrupt
System Interval Timer
Soft Error Log
C170 Exchange Request

(iii) **User Condition**

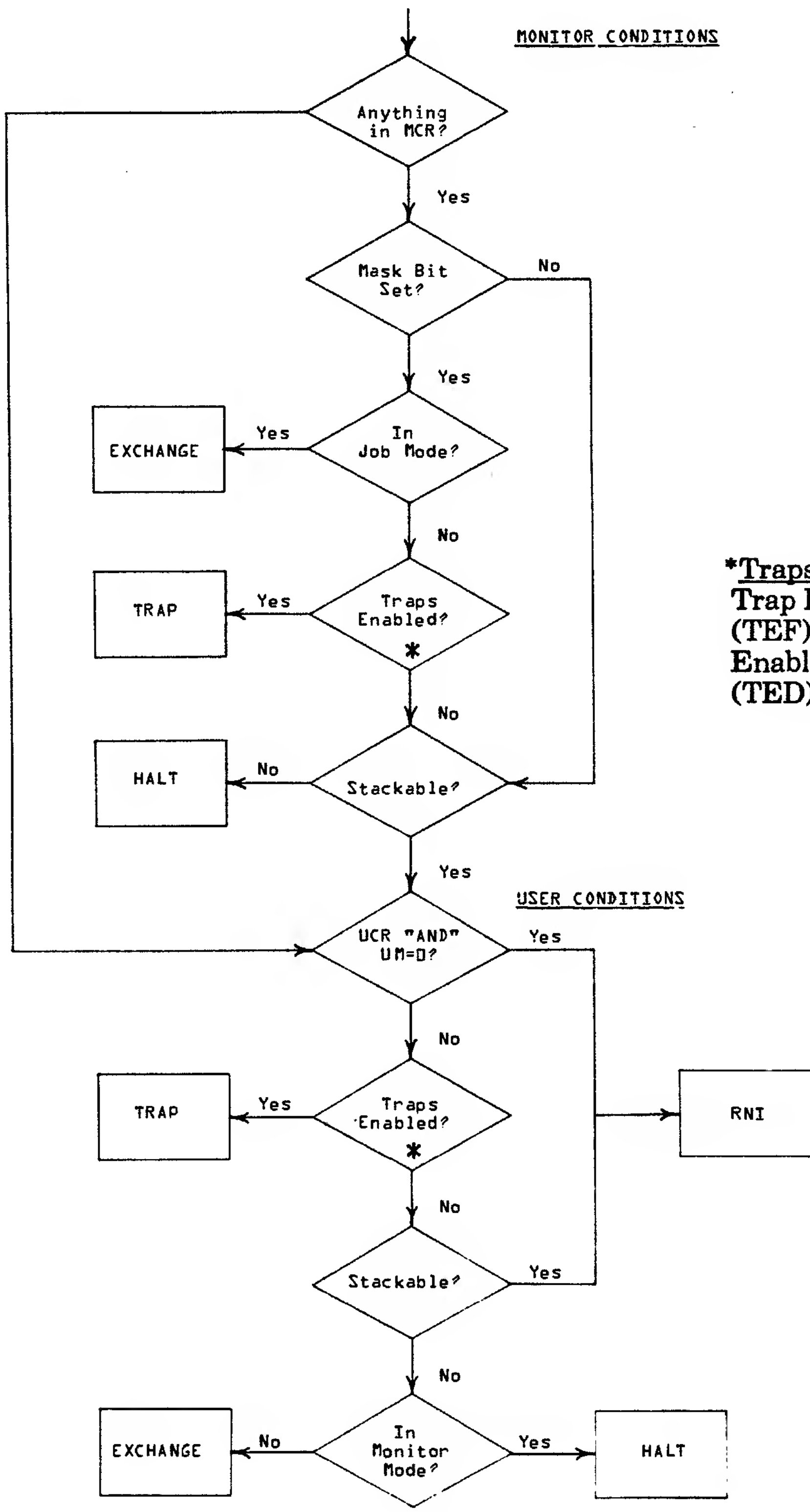
User conditions are exception conditions directly related to the active process which do not preclude further processing. They are:

Free Flag
Process Interval Timer
Keypoint
Divide Fault
Debug
Arithmetic Overflow
Exponent Overflow
Exponent Underflow
Floating Point Loss of Significance
Floating Point Indefinite
Arithmetic Loss of Significance
Invalid BDP Data

(iv) **Status Indicators**

These indicators do not give rise to interrupt conditions, but are set to enable software (monitor) to determine what action to take. They are:

System Call
Trap Exception



*Traps Enabled means Trap Enable Flip-flop (TEF) set and Trap Enable Delay Flip-flop (TED) clear.

Figure 2.8-1. Interrupt Flowchart

2.8.10 Flags

The state of the five flags: On Condition Flag (OCF), Critical Frame Flag (CFF), Keypoint Enable Flag (KEF), Trap Enable Flip-flop (TEF) and Trap Enable Delay Flip-flop (TED), after the completions of the operations: CALL, RETURN, POP, EXCHANGE and TRAP shall be as indicated in the table below:

	CFF	OCF	KEF	TEF	TED
CALL	C	C	A	A	A
RETURN	PS	PS	A	A	C
POP	PS	PS	A	A	A
EXCHANGE	XP	XP	XP	XP	XP
TRAP	C	C	A	C	A

Legend:
- C - Cleared by operation
- A - As is (unchanged by operation)
- PS - Loaded by operation from previous stack frame save area
- XP - Loaded by operation from exchange package

2.9 BUFFERS

Up to three buffers to increase processor performance may be included in the processor. These buffers are described in the following sections. The existence, size, performance, and organization of these buffers shall be processor model-dependent.

2.9.1 Map Buffer

The MAP Buffer shall be a high speed memory used to eliminate repeated references to the segment tables and the page table. Map size, operation and entry replacement algorithm shall be processor model-dependent.

The processor may assume that valid entries in the Segment Table and/or Page Table have not been changed to differ from the corresponding entry in the MAP and thus, the processor need not verify a MAP entry with the Segment Table and/or Page Table. The processor shall not make any entries in the MAP which are derived from invalid Page Table entries. The processor may make entries in the Map which are derived from invalid Segment Table entries or from entries beyond the Segment Table Length providing that attempts to use these entries result in an Invalid Segment exception condition.

The processor shall be allowed to purge entries from the MAP at any time.

No entry based on a Segment Descriptor shall be made into the MAP unless the entry is the result of the current or known future instruction. (Thus, entries to support lookahead procedures are acceptable but random entries such as loading a target Segment Descriptor plus the next one or several is not acceptable.)

The processor may make entries into the MAP based on Page Table entries which are not related to the current or future instruction. (Thus, random entries such as loading a target Page Table entry plus the next several are acceptable.)

2.9.2 Cache Buffer

The Cache Buffer shall be a high speed memory which shall be used to reduce the access time to central memory for words which are used more than once. The entries in cache shall be based on System Virtual Addresses (SVAs).

Cache size, operation, and entry replacement algorithm will be processor model-

dependent. However, every instruction which modifies (stores into) central memory shall issue the appropriate request(s) on the central memory interface, irrespective of any associated, model-dependent cache operations.

The processor may assume that the data in central memory has not been changed to differ from the corresponding entries in cache and thus, the processor need not verify the cache entry with central memory on read operations with non-cache-bypass segments. The processor shall appropriately maintain the integrity of cache on processor generated writes to non-cache-bypass segments in central memory *and on IOU generated 60-bit writes (7.2.5.4).* It is <u>not</u> a hardware responsibility to purge cache when central memory is altered by some method other than by a processor generated write to a non-cache-bypass segment *or an IOU generated 60-bit write.*

The processor shall be allowed to purge entries from the cache at any time.

2.9.3 Instruction Stack

The Instruction Stack is any memory, register, pipeline, etc. used to hold instructions apart from the central memory and caches to the central memory. (Processors having caches which contain instructions shall appropriately update or purge the instructions in cache upon central memory writes as per 2.9.2.) The instruction stack size, operation and management algorithm is model-independent.

The instruction stack shall be purged, at least, at the following points:

C180 Exchange Operation
AND
Intersegment Branch or Purge Buffer with k=4

The future hardware development projects are free to implement the purge on either the Intersegment Branch or the Purge Buffer instruction and must always purge on the C180 Exchange. See 1.6 for a list of specific model implementations.

2.9.4 Stack Frame Save Area (SFSA) Pushdown

SFSA Pushdown is a model-dependent feature. See section 1.6 for a list of the C180 systems supporting SFSA Pushdown.

SFSA Pushdown shall be a high speed memory which allows a processor to store a portion of the SFSA created by a CALL operation for subsequent retrieval by the RETURN operation.

2.9.4.1 General Operation of SFSA Pushdown

Processors having the SFSA Pushdown (see section 1.6) retain a copy of the P register and the SFSA Descriptor in the SFSA Pushdown as well as store the entire SFSA into the normally devined area in central memory.

While the processor is free to purge the contents of the SFSA Pushdown at any time, there are several situations which shall cause the processor to void the contents of the SFSA Pushdown:

- Any job to monitor exchange.
- Any half exchange.
- Execution of the 07 Purge SFSA Pushdown instruction with the sub-ob k=1.
- Trap operation.

The processor must handle the case of the SFSA Pushdown being full on a CALL or TRAP operation. Acceptable solutions on a model-dependent basis include but are not limited to the following:

- Moving the oldest SFSA out of the SFSA Pushdown into its defined location in the stack in central memory.
- Moving a model dependent number of the older SFSAs into their defined locations in central memory when the SFSA Pushdown reaches some threshold.
- Moving the entire SFSA Pushdown into the stack in central memory.

The number of SFSAs retained in the SFSA Pushdown is model dependent.

The SFSA Pushdown may but need not be implemented in monitor mode.

The processor may assume that the SFSA Pushdown is empty when executing a C180 monitor to job exchange operation. Any attempt to execute a monitor to job exchange with a SFSA defined in the stack for monitor will result in undefined operation.

When a SFSA or portion thereof is obtained from the SFSA Pushdown on a RETURN instruction, the hardware is not required to perform all of the exception tests which would be required if the SFSA came from the central memory. See the RETURN instruction for more detail.

2.10 INTERFACES

2.10.1 Central Memory

The processor central memory interface shall be compatible with the central memory interface specified in 4.1.3 and 4.2 of this specification. Compatible shall mean that all signals and operations shall be provided as specified in 4.1.3 and 4.2 except that transmitted signals become received signals and vice versa.

2.10.1.1 Processor Central Memory Port Selection

- Processors having an External Processor Port (see 1.3.3) shall perform as follows:
  - a. When these two ports are connected to independent memories, as illustrated in figure 1.3-5 of this specification, the processor central memory port used for any given central memory access shall be determined by the state of bit 33 of the Real Memory Address, (see 3.1.3), as used for the central memory access. If bit 33 is clear, the Local Processor Port to the processor's own central memory is selected. If bit 33 is set, the External Processor Port to a central memory within another system is selected.
  - b. When only a single port is present (the Local Processor Port) the processor need not detect and take special action for any reference with bit 33 set (External Processor port) except as described in paragraph 2.5.1.11, but may let the reference continue which will result in a time-out (8.2.9) and, thus, a detected malfunction.
- Processors not having an External Processor Port (see 1.3.3) shall ignore bit 33 of the RMA and direct the reference to the processor's central memory. These processors shall ignore the Shared Memory Present bit in the DEC.

2.10.2 Maintenance Access

The maintenance access interface is model-dependent and shall be documented in the appropriate model-dependent engineering specification. See section 6.0 for the model-independent characteristics that are required.

2.11 PERFORMANCE MONITORING FACILITY (PMF)

The PMF need not be identical for the different processors; however, the operation of the PMF shall be model-independent except for the length of the major clock cycle and specific events or states as noted in paragraph 2.11.5.

The basic requirements for the PMF are as follows:

a. Performance measurements on a dual processor mainframe require two PMFs.

b. The PMF shall be controlled and accessed for data via Maintenance Access Reads and Writes of register 22. When PMF is not installed, register 22 reads and writes shall be performed as abnormal requests to a nonexistent register.

c. The PMF shall provide eight 32-bit counters capable of monitoring the specified events and/or states with no performance impact upon the associated processor.

d. The PMF shall contain one register as shown in figure 2.11-1.

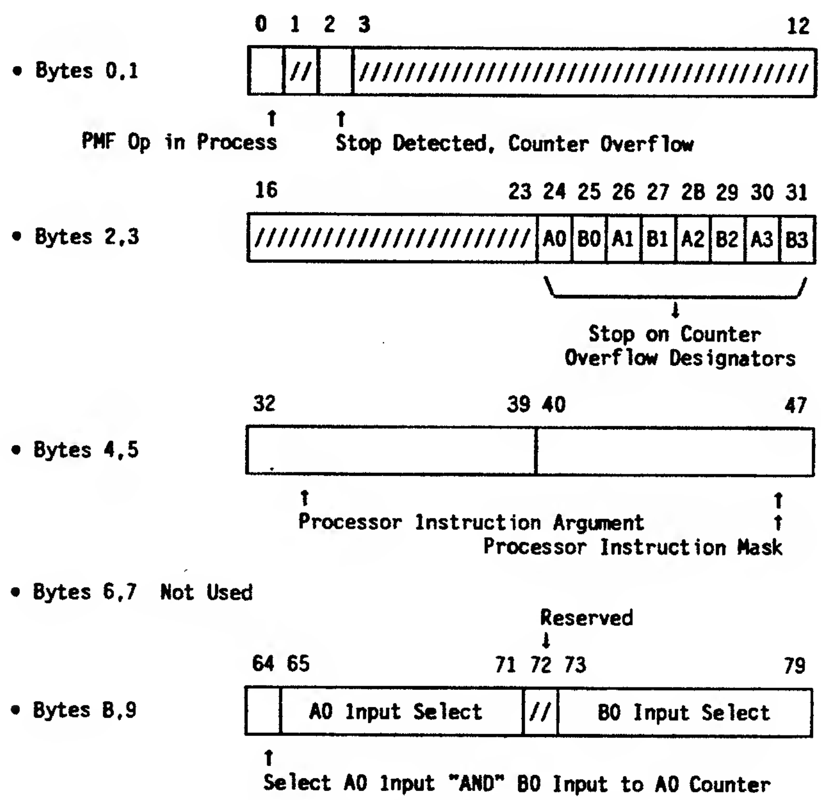
   Register 22 controls the PMF operation and contains the eight 32-bit counters.

   See 2.6.5.2 for a description of the Maintenance Register

See paragraph 1.6 for more information.

Performance monitoring on processors not having a PMF shall be done in accordance with paragraph 8.4.

REGISTER 22



- Bytes 10,11 A1, B1 Input Select (same format as bytes 8, 9)
- Bytes 12,13 A2, B2 Input Select (same format as bytes 8, 9)
- Bytes 14,15 A3, B3 Input Select (same format as bytes 8, 9)
- Bytes 16-19 Counter A0
- Bytes 20-23 Counter B0
- Bytes 24-27 Counter A1
- Bytes 28-31 Counter B1
- Bytes 32-35 Counter A2
- Bytes 36-39 Counter B2
- Bytes 40-43 Counter A3
- Bytes 44-47 Counter B3

Figure 2.11-1. PMF Register Formats

2.11.1 PMF Initialization/Operation

PMF Initialization shall be defined as the writing of all 48 bytes of processor register 22 via the Maintenance Access. The writing of the first byte shall cause the PMF to do the following:

- Halt any current PMF operation
- Clear byte 0 of register 22 (2.11.2)

After all 48 bytes have been written, the PMF shall start operation. Any write of register 22 which is less than 48 bytes shall place the PMF in an inactive state.

Maintenance Access Read requests for register 22 after PMF Initialization and before the end of PMF operation shall result in a DISCONNECT being sent to the IOU via the Maintenance Access immediately upon the second request. The entire 48 bytes of register 22 may be read via the Maintenance Access after the end of PMF operation.

PMF operation is that time period during which counts may be made of various events and states within the processor during PMF operation. PMF operation shall be terminated upon the occurrence of either of the following events:

- Stop on Counter Overflow detected (bits 24-31 of register 22)
- Any write into register 22 via the Maintenance Access.

There shall be only one period of PMF operation following a PMF Initialization.

2.11.2 PMF Status (Register 22 Byte 0)

Byte 0 of register 22 contains the PMF Status bits. A Maintenance Access Write of this byte shall cause all 8 bits to be cleared regardless of the write data from the channel. (Thus, all PMF Status bits are cleared during initialization.) Bit 2, once set, will remain set until the next PMF initialization.

Bit 0 - PMF Operation in Process

Bit 0 shall set when a PMF Operation (as defined in 2.11.1) begins and shall clear when the PMF Operation terminates.

Bit 2 - STOP Detected, Counter Overflow

Bit 2 shall set whenever a PMF Operation is terminated because of a counter overflow as specified by bits 24 through 31 of register 22.

Bits 1, 3-7 - Not Assigned

These bits shall be zero when read via the Maintenance Access. Writes into these bits shall be ignored. (Refer to 2.1.3.5b.)

2.11.3 PMF Control (Register 22 Bytes 1-7)

Byte 3 contains counter overflow selections. Bytes 4 and 5 contain Instruction Argument and Mask for event code 0C. Bytes 1, 2, 6 and 7 are not used.

2.11.3.1 PMF Control Bits (Bytes 1-3)

The bits in byte 1 shall control the PMF Operation as defined below:

Bits 8-23 - Not Assigned

The processor shall ignore the state of these bits. The writing of any value other than zero into these bits shall cause undefined operation as described in 2.1.3.5b. These bits shall be zero when read assuming the previous write was zero.

Bits 24-31 - Counter Overflow Step

Any of these eight bits, when set during PMF initialization, shall cause the PMF Operation to terminate immediately upon the overflow (carry out of leftmost bit position) of the corresponding 32-bit counter.

Bit Number:	24	25	26	27	28	29	30	31
Counter:	A0	B0	A1	B1	A2	B2	A3	B3

2.11.3.2 PMF Control (Bytes 4-7)

Bytes 4 and 5 either contain instruction Argument and Mask or are not used depending on the implementation of instruction counting. Bytes 6 and 7 are never used. The writing of any values other than zero into these bytes shall cause undefined operation as described in 2.1.3.5b. The unused bytes shall be zero when read by the Maintenance Access assuming the previous write was zero.

2.11.4 PMF Counters (Register 22 Bytes 8-47)

The occurrence of certain events or states (2.11.5) within the processor shall be available to the PMF for its use in accumulating individually selectable counts in its eight 32-bit counters (see figure 2.11-2). These counters shall be divided into two groups of four counters each, and shall be designated A0 through A3 and B0 through B3.

The contents of these counters may be read as bytes 16 through 47 of register 22 (see figure 2.11-1). Moreover, these counters may be initialized to predetermined values via the Maintenance Access Write of bytes 16 through 47 as part of PMF Initialization (2.11.1).

The input to each of these counters shall be individually selected as specified in bytes 8 through 15 of register 22. These bytes shall be formatted as shown in figure 2.11-1. The code for each input selector shall be up to 7 bits (model-dependent) and shall select an input event or state as specified in paragraph 2.11.5. The engineering specifications for each processor shall clearly define these codes. The reserved bits shall be ignored by the processors and undefined on read operations.

The writing of any value other than zero into the unassigned bits in bytes 8 through 15 shall cause undefined operation with respect to the subsequent value of these bits. These bits shall be zero when read assuming the previous write was zero as noted in 2.1.3.5b.

Bit 0 of bytes 8, 10, 12 and 14, when clear, shall cause the appropriate A counter to receive the selected A input. Bit 0 of bytes 8, 10, 12 and 14, when set, shall cause the appropriate A counter to receive the "AND" of the selcted A and B inputs. For example, bit 0 of byte 8 shall cause counter A0 to receive the A0 input selection "AND" the B0 input selection (see table 2.11-1). Whether or not bit 0 is set, the appropriate B counter shall receive the selected B input.

Events when selected to a counter shall cause the counter to increment by one each time the event occurs. States are conditions from the processor (such as Monitor Mode) which last for more than one clock period. States when enabled to the A counters shall cause the counter to increment once each time the state occurs and when enabled to the B counters shall cause the counter to increment once each clock period that the state exists.

These counters may be used to stop PMF operation upon counter overflow as described in 2.11.3.1. The PMF shall be allowed to run for an additional 0 to 47_{10} clocks after the overflow occurs before the stop PMF operation blocks additional event counting. Thus, counters which were initialized to zero and have a value in the range of 0 to 47_{10} after a stop PMF operation due to overflow may be ambiguous with respect to whether or not they overflowed. For this reason it is recommended that all counters be initialized with a value greater than 48_{10}.

All counters must stop event counting at the same clock no matter which value in the range 0 to 47_{10} that stop occurs.

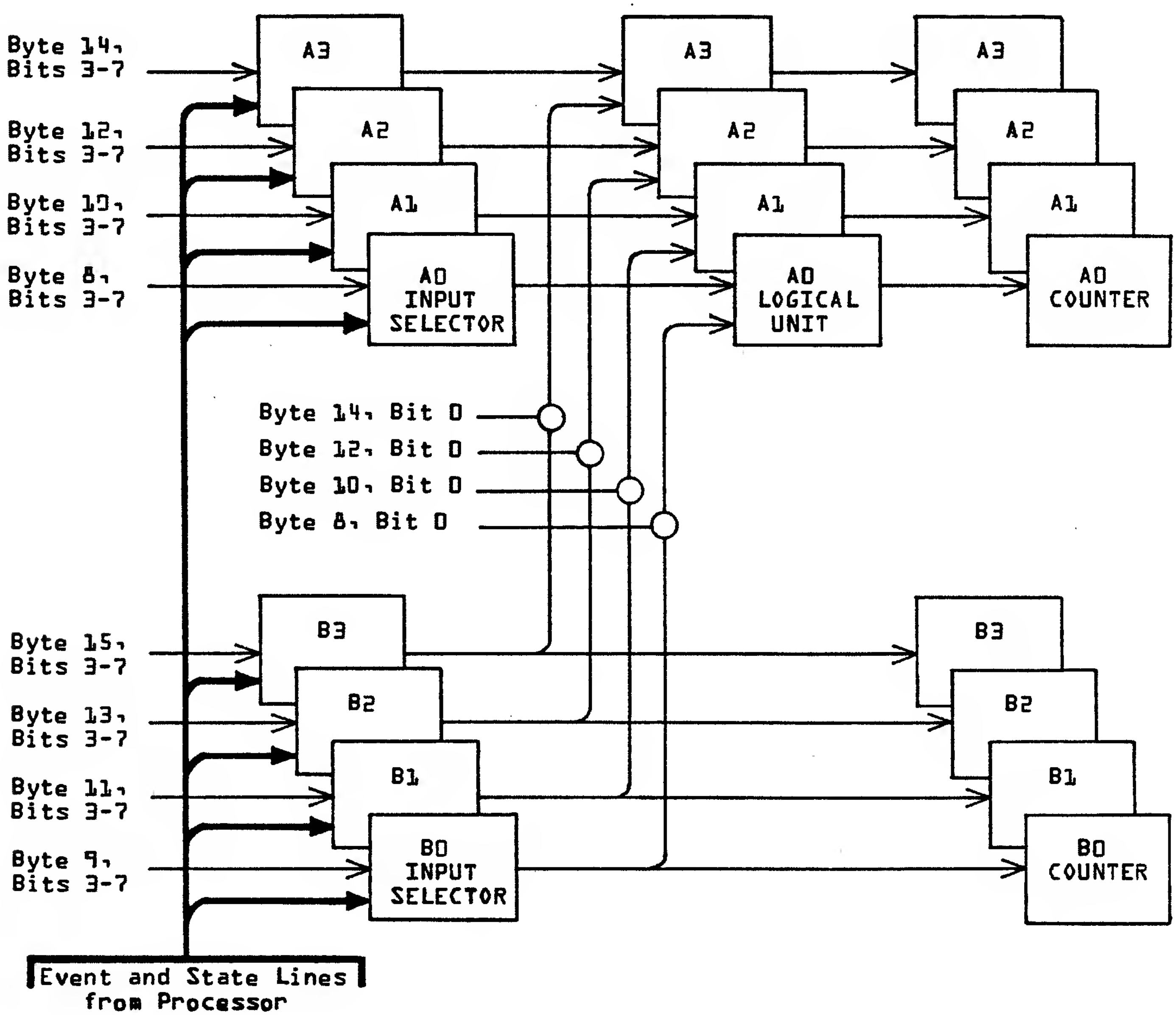


Figure 2.11-2. PMF Input Selectors and Counters

INPUT SELECTORS		COUNTERS		
		Byte n, Bit 0 Clear		///////////////
		///////////////	Byte n, Bit 0 Set	
A	B	A	B	A
Event X	Event Y	Count of Event X	Count of Event Y	Undefined
Event X	State Y	Count of Event X	Count of time in State Y*	Occurrences of Event X while State Y exists
State X	Event Y	Count of occurrences of State X	Count of Event Y	Occurrences of Event Y while State X exists
State X	State Y	Count of occurrences of State X	Count of time in State Y*	Undefined

* See Engineering Specifications for length of clock cycles.

Table 2.11-1. Definition of PMF Counter Actions

2.11.5 Events and States

Each processor shall support measurement of the events, states and functions described below. Functions are descriptions of the required measurement but not in terms of specific events. The ability to measure these functions is a model-independent requirement, however, the specific events used to obtain the measurement are model-dependent. The engineering specifications shall describe the method of measuring these functions; i.e., which events or states must be measured and appropriately combined to obtain the function. The engineering specifications shall also list and describe each event and state provided.

Each processor shall support the following list of events and states as a minimum, however, this list is a minimum set and it is expected that each processor may implement additional events and states appropriate to model-dependent design analysis.

Events

- One microsecond

  This event shall occur once each microsecond.

- Page Faults

  This event shall occur once each time a Page Table Search Without Find occurs.

- Count of Specific Instructions

  This event shall allow counting the number of executions of a specific instruction or group of instructions. These instructions are selected prior to the measurement period. Bits 32 through 47 of PMF register 22 are reserved to support the selection of instructions. Instructions may also be selected by flags in the microcode.

- Instruction Complete

  This event shall occur once each time an instruction completes execution.

States

- C180 Monitor Mode

  This state line shall be a one whenever the processor is in C180 Monitor Mode (2.5.1.11) and shall be a zero whenever the processor is not in C180 Monitor Mode. This line shall make one, and only one, transition from one to zero (or zero to one) during each C180 exchange operation.

- C180 Virtual State

  This state line shall be a one whenever the processor is in C180 Virtual State. An exchange from C180 Job to Monitor, etc. shall not cause a transition on this line.

- Trap Enabled

  This state line shall be a one whenever the processor has Traps Enabled and shall be a zero whenever the processor has Traps Disabled as defined in 2.8.

- *C170 State*

  This state line shall be a one whenever the processor is in C170 State. (The logical inverse of C180 virtual state is satisfactory.) A C170 exchange from C170 Job to C170 Monitor, etc. shall not cause a transition on this line.

Functions

- Segment MAP

  This is a measurement of the additional execution time required due to misses in the Segment MAP. An event which allows counting the number of additional clocks required in instruction execution due to Segment MAP misses would provide the most accurate measure. It is acceptable to count the number of Segment MAP misses and provide an average calculated time impact (which may then be multiplied by the number of misses).

- Page MAP

  This is a measurement of the additional execution time required due to misses in the Page MAP. An event which allows counting the number of additional clocks required in instruction execution due to Page MAP misses would provide the most accurate measure. It is acceptable to count the number of Page MAP misses and Page Table entries searched and provide a method to approximate execution impact due to Page MAP misses.

- Cache Hit Ratio for Operands

  This is a measurement of the number of read references to cache for operands and the number of these references which result in a cache hit. There are many sets of events which could be chosen to provide this measurement.

2.12 VECTOR INSTRUCTIONS

The vector instructions (table 2.12-1) are, in general, three-address, memory to memory vector operations which are available on certain CYBER 180 systems as specified in section 1.6. These instruction op codes shall be detected as Unimplemented Instructions on CYBER 180 systems other than those specified.

2.12.1 General Description

2.12.1.1 Format

All vector instructions utilize the jkiD instruction format. Designators j and k always designate the register Aj and Ak where (Aj) points to the starting address of a source vector, VAj, and (Ak) points to the starting address of the destination vector, VAk. Designator i typically designates register Ai where (Ai) points to the starting address of a second source vector, VAi. The exceptions (convert vectors, gather/scatter vectors) where i is used in a different manner are described in the individual instruction descriptions. An Address Specification error shall be recorded whenever the rightmost three bits of Aj, Ak or Ai (when used) are not all zeros.

The second bit from the left in the D field shall be ignored and should be set to zero.

The following instructions are the standard vector instructions (see 1.6):

INSTRUCTION NAME	OP. CODE	MNEMONIC
Integer Vector Sum	44jkiD	ADDXV
Integer Vector Difference	45jkiD	SUBXV
Integer Vector Compare, =	50jkiD	CMPEQV
Integer Vector Compare, <	51jkiD	CMPLEV
Integer Vector Compare, ≥	52jkiD	CMPGEV
Integer Vector Compare, ≠	53jkiD	CMPNEV
Shift Vector Circular	4DjkiD	SHFV
Logical Vector Sum	48jkiD	IORV
Logical Vector Difference	49jkiD	XORV
Logical Vector Product	4AjkiD	ANDV
Convert Vector from Int. to F.P.	4BjkiD	CNIFV
Convert Vector from F.P. to Int.	4CjkiD	CNFIV
Floating Point Vector Sum	40jkiD	ADDFV
Floating Point Vector Difference	41jkiD	SUBFV
Floating Point Vector Product	42jkiD	MULFV
Floating Point Vector Quotient	43jkiD	DIVFV
Floating Point Vector Summation	57jkiD	SUMFV
Merge Vector	54jkiD	MRGV
Gather Vector	55jkiD	GTHV
Scatter Vector	56jkiD	SCTV

The following instructions are the extended vector instructions (see 1.6):

INSTRUCTION NAME	OP. CODE	MNEMONIC
Floating Point Vector Triad, * +	58jkiD	TPSFV
Floating Point Vector Triad, * -	59jkiD	TPDFV
Floating Point Vector Triad, + *	5AjkiD	TSPFV
Floating Point Vector Triad, - *	5BjkiD	TDPFV
Floating Point Vector Dot Product	5CjkiD	SUMPFV
Gather Vector, Index	5DjkiD	GTHIV
Scatter Vector, Index	5EjkiD	SCTIV

Table 2.12-1. Vector Instructions

2.12.1.2 Length (Number of Operations)

Models using standard vectors (see 1.6)

The rightmost ten bits of the D field, when nonzero, specify the length or number of operations to be performed (1 to 512). When the rightmost ten bits of the D field are zero, then the length is specified by X1 Right. When X1 Right is negative, an Instruction Specification Error shall be recorded. When X1 Right is positive and less than 512_{10}, then this number (from X1 Right) shall be used as the length for the vector instruction. When X1 Right is greater than or equal to 512_{10}, then 512_{10} shall be used as the length for the vector instruction. An Instruction Specification Error shall be recorded when the rightmost ten bits of D are greater than 512_{10}.

When the rightmost ten bits of D and all 32 bits of X1 Right are zero, the instruction shall be performed as described in paragraph 2.1.7.

Models using standard and extended vectors (see 1.6)

The rightmost ten bits of the D field, when nonzero, specify the length or number of operations to be performed (1 to 512). When the rightmost ten bits of the D field are zero, then the length is specified by X1 Right. When X1 Right is positive and less than 512_{10}, then this number (from X1 Right) shall be used as the length for the vector instruction. When X1 Right is greater than or equal to 512_{10}, then 512_{10} shall be used as the length for the vector instruction. An Instruction Specification Error shall be recorded when the rightmost ten bits of D are greater than 512_{10}.

When the rightmost ten bits of D are zero and X1 Right is negative or zero, the instruction shall be performed as described in paragraph 2.1.7.

2.12.1.3 Broadcast

The leftmost bit of the D field, when set, shall cause VAj to be generated by repeating the single element contained in Xj for all vector instructions.

2.12.1.4 Interrupts

The interrupt response time shall be less than 20 microseconds for all instructions.

- All Vector Instructions other than Gather/Scatter

  These vector instructions are not interruptable after any results have been stored into central memory. When a group 2A condition bit (table 2.8-3) sets in the MCR or UCR that specifies a program interruption (tables 2.8-1 and 2.8-2) before results are stored into central memory, the instruction is inhibited and appears conceptually not to have executed. When a group 2A condition bit sets after any results are stored into central memory, the instruction execution is completed before any program interrupt is initiated.

- Gather/Scatter Instructions

  The gather/scatter instructions may be interrupted when a group 2A condition bit sets in the MCR or UCR after results have been partially stored into central memory as described in paragraph 2.12.10.

2.12.1.5 Results (Scalar/Vector)

When a vector instruction performs an operation for which a comparable scalar instruction exists, the vector result shall be identical to the result obtained on the scalar instruction using the same input operands. Tables 2.12-2(A&B) specify the comparable operations for all vector instructions except for Summation, Merge, Triad, Gather and Scatter. (These have no comparable scalar operation.)

There are four exception conditions for which scalar instructions inhibit the store operation when traps are enabled and the associated mask bit is set. The result to be stored by vector instructions when the comparable scalar operation does not store a result.

- Divide Fault - UCR55

  The vector operation (Op. 43) will store an Indefinite result (700...0) whenever a Divide Fault is detected (as noted in tables 2.4-9 and 2.4-10), and both a Divide Fault and a FP Indefinite condition will be detected.

- Arithmetic Loss of Significance - UCR60
- Floating Point Indefinite - UCR61
- Arithmetic Overflow - UCR62

  For these latter three conditions, the vector instructions shall store the specified result, as per the appropriate user mask bit, which the scalar instruction would have stored when the traps are disabled.

	VAj is a source vector of contiguous	VAk is destination vector of contiguous	VAi is a source vector of contiguous	Comparable Scalar Operation
Integer Sum Integer Difference	64-bit integers	64-bit integers	64-bit integers	2.2.2.1 2.2.2.2
Integer Compare = Integer Compare < Integer Compare ≥ Integer Compare ≠	64-bit integers	64-bit elements	64-bit integers	2.2.2.9
Shift Circular	64-bit elements	64-bit elements	64-bit elements	2.2.7.1
Logical Sum Logical Difference Logical Product	64-bit elements	64-bit elements	64-bit elements	2.2.8.1
Convert fm Int to FP Convert fm FP to Int	64-bit integers F.P. operands	F.P. operands 64-bit integers	Not Used Not Used	2.4.2.1 2.4.2.2
F.P. Sum F.P. Difference F.P. Product F.P. Quotient	F.P. operands	F.P. operands	F.P. operands	2.4.3.1 2.4.3.1 2.4.3.2 2.4.3.3
F.P. Summation	Not Used	One F.P. operand stored into Xk rather than into central memory at VAk	F.P. operands	None
Merge	64-bit elements	64-bit elements	64-bit elements	None
Gather	VAj is source vector of typically discontiguous 64-bit elements	VAk is destination vector of contiguous 64-bit elements	Xi contains the interval	None
Scatter	VAj is source vector of contiguous 64-bit elements	VAk is destination vector of typically discontiguous 64-bit elements	Xi contains the interval	None

Note: See section 1.6 for a list of systems using standard vector instructions.

Table 2.12-2A. Standard Vector Instruction Input and Output Fields

	VAj is a source vector of contiguous	VAk is destination vector of contiguous	VAi is a source vector of contiguous	Comparable Scalar Operation
F.P. Triad, * + F.P. Triad, * - F.P. Triad, + * F.P. Triad, - *	F.P. operands	F.P. operands	F.P. operands	None
F.P. Dot Product	F.P. operands	One F.P. operand stored into Xk rather than into central memory at VAk	F.P. operands	None
Gather, Index	VAj is source vector of typically discontiguous 64-bit elements	VAk is destination vector of contiguous 64-bit elements	VAi is source vector of contiguous 64-bit indexes	None
Scatter, Index	VAj is source vector of contiguous 64-bit elements	VAk is destination vector of typically discontiguous 64-bit elements	VAi is source vector of contiguous 64-bit indexes	None

Note: See section 1.6 for a list of systems using extended vector instructions.

Table 2.12-2B. Extended Vector Instruction Input and Output Fields

2.12.1.6 Condition Register Bits

- DUE

  The above condition register bit is as defined in paragraph 2.8.1.1. (The PVA contained in P does not necessarily point to the instruction which initiated the activity resulting in this malfunction.)

- Instruction Specification Error

  Environment Specification Error
  Invalid Segment
  Access Violation
  Address Specification Error (see Note below)
  Page Table Search without Find (see Note below)

  The above condition register bits (for all vector instructions except gather/scatter), when applicable as shown in appendix D, shall cause the instruction execution to be inhibited and the appropriate interrupt to be taken as specified in tables 2.8-1 and 2.8-2.

  Note: The Address Specification Error and Page Table Search without Find conditions need not cause the execution of the gather/scatter

  instructions to be inhibited but rather to be halted as described in 2.12.10.

  The PVA contained in P at the time the above interrupt occurs shall point to the vector instruction which caused the interrupt.

- Debug

  The Debug condition register bit applies to all vectors. The address of the first word of each source vector Aj and Ai (when present) shall be compared for read data. The address of the first word of the destination vector Ak shall be compared for write. The detection of a debug condition shall cause the instruction execution to be inhibited and the trap operation to be performed. The PVA contained in P at the time of the debug interrupt shall point to the vector instruction which caused the debug interrupt to occur.

- Divide Fault

  Arithmetic Overflow
  Exponent Overflow
  Exponent Underflow
  F.P. Loss of Significance
  F.P. Indefinite
  Arithmetic Loss of Significance

  The above condition register bits, where applicable as shown in appendix D, are detected and set in the UCR at the completion of the vector instruction. These bits are, in effect, the "OR" of multiple operations. The instruction execution is not inhibited and the interrupt specified in table 2.8-2 occurs after the completion of the vector instruction. The PVA contained in P at the time the above interrupt occurs shall point to the instruction following the vector instruction that contained the operation(s) which caused the interrupt condition bit(s) to be set if an Exponent Overflow, Exponent Underflow, or Floating Point Loss of Significance condition occurred and the associated mask bit is set.

  Otherwise, the PVA contained in P shall point to the vector instruction which contained the operation(s) which caused the interrupt condition bit(s) to be set. Note that when multiple interrupt conditions occur that indicate different values of P, then P points to the instruction following the one that contained the operation(s) which caused the interrupt condition bit(s) to be set.

2.12.1.7 Overlap

Source and destination vectors for the same instruction may be overlapped only when the starting address of the destination vector is less than or equal to the starting address of the source vector.

All other cases of overlap of source and destination vectors within a single instruction shall be undefined with respect to the results.

2.12.1.8 Page Size

The page size shall be 4096 bytes or larger when executing any vector instruction. When a vector op code is encountered (on a processor with vectors implemented) and the page size is less than 4096 bytes, an Environmental Specification Error shall be recorded and the execution of the vector instruction shall be inhibited.

It must be noted that, while other vectors require no more than two pages each, the input vector for the gather instruction and the output vector for the scatter instruction require an increasing number of pages to be present in central memory as the interval increases. The maximum number of pages, 512_{10}, is required when the interval is equal to or larger than the page size. A gather or scatter instruction limited to an insufficient number of pages could continually interrupt with the Page Table Search without Find condition bit set, and never complete execution. The page management algorithm of the operating system must take this into account.

2.12.2 Integer Vectors - Arithmetic

a. Integer vector sum, V(Ak) replaced by V(Aj) plus V(Ai)

44jkiD (Ref. 172)

b. Integer vector difference, V(Ak) replaced by V(Aj) minus V(Ai)

45jkiD (Ref. 173)

These instructions shall perform the indicated arithmetic operation on the first element from V(Aj) and V(Ai) and store the result as the first element of V(Ak). This operation is repeated for successive elements until the required number of operations has been performed.

2.12.3 Integer Vectors - Compare

a. Integer vector compare, V(Ak) replaced by V(Aj) equal to V(Ai)

50jkiD (Ref. 176)

b. Integer vector compare, V(Ak) replaced by V(Aj) less than V(Ai)

51jkiD (Ref. 177)

c. Integer vector compare, V(Ak) replaced by V(Aj) greater than or equal to V(Ai)

52jkiD (Ref. 178)

d. Integer vector compare, V(Ak) replaced by V(Aj) not equal to V(Ai)

53jkiD (Ref. 179)

These instructions shall perform the indicated integer arithmetic comparison on the first element from V(Aj) and V(Ai). If the compare is true, bit 0 is set and bit positions 1 through 63 are cleared in the first element of V(Ak). If the compare is false, bit positions 0 through 63 are cleared in the first element of V(Ak). This operation is repeated for successive elements until the number of required comparisons has been performed. When broadcast of V(Aj) is selected and j=0, the contents of the X0 Register shall be interpreted as consisting entirely of all zeros.

2.12.4 Shift Vector Circular

Shift vector circular, V(Ak) replaced by V(Ai), direction and count per V(Aj)

4DjkiD (Ref. 180)

This instruction shall perform a circular shift on the first element from V(Ai) as directed by the first element of V(Aj) and store the result as the first element of V(Ak). This operation is repeated for successive elements until the required number of operations has been performed. The shift count for each element in V(Ai) is taken from the rightmost 8 bits of the corresponding element of V(Aj) and interpreted as described below.

The sign bit in the leftmost position of the 8-bit shift count shall determine the direction of the shift. When the shift count is positive (sign bit of zero), these instructions shall left shift. When the shift count is negative (sign bit of one), these instructions shall right shift. Shifts shall be from 0-63 bits left and from 1-64 bits right. Based on an 8-bit signed two's complement shift count, these shifts are as follows:

```
0111 1111 }
    :     } Left Shift 0-63
    :     }
0100 0000 }
0011 1111   Left Shift 63
    :            :
    :            :
0000 0000   Left Shift 0
---------------------------
1111 1111   Right Shift 1
    :            :
    :            :
1100 0000   Right Shift 64
1011 1111 }
    :     } Right Shift 1-64
    :     }
1000 0000 }
```

When these interpretations of the shift count result in an actual shift count of zero, the associated instructions shall transfer the initial element of V(Ai) to the corresponding element in V(Ak) with no shift.

When broadcast of V(Aj) is selected and j=0, the contents of the X0 Register shall be interpreted as consisting entirely of zeros.

2.12.5 Logical Vectors

a. Logical vector sum, V(Ak) replaced by V(Aj) OR V(Ai)

48jkiD (Ref. 181)

b. Logical vector difference, V(Ak) replaced by V(Aj) ~~EØR~~ XOR V(Ai)

49jkiD (Ref. 182)

c. Logical vector product, V(Ak) replaced by V(Aj) AND V(Ai)

4AjkiD (Ref. 183)

These instructions shall perform the indicated logical operation on the first word from V(Aj) and V(Ai) and store the result as the first word of V(Ak). This operation is repeated for successive elements until the required number of word logical operations has been performed.

2.12.6 Convert Vectors

a. Convert vector, floating point V(Ak) formed from integer V(Aj)

4BjkiD (Ref. 184)

b. Convert vector, integer V(Ak) formed from floating point V(Aj)

4CjkiD (Ref. 185)

These instructions shall perform the indicated convert operation on the first element from V(Aj) and store the result as the first element of V(Ak). This operation is repeated for successive elements until the required number of convert operations have been performed. Designator i is ignored by these instructions.

2.12.7 Floating Point Vectors - Arithmetic

a. Floating point vector sum, V(Ak) replaced by V(Aj) plus V(Ai)

40jkiD (Ref. 186)

b. Floating point vector difference, V(Ak) replaced by V(Aj) minus V(Ai)

41jkiD (Ref. 187)

c. Floating point vector product, V(Ak) replaced by V(Aj) times V(Ai)

42jkiD (Ref. 188)

d. Floating point vector quotient, V(Ak) replaced by V(Aj) divided by V(Ai)

43jkiD (Ref. 189)

These instructions shall perform the indicated arithmetic operations on the first element from V(Aj) and V(Ai) and store the result as the first element of V(Ak). This operation is repeated for successive elements until the required number of operations has been performed.

2.12.8 Floating Point Vector Summation

Floating Point Vector Summation, Xk replaced by summation of elements in V(Ai)

57jkiD (Ref. 190)

This instruction shall add together all of the elements in V(Ai) and store that sum in Xk. The individual add operations which together form this instruction are single precision sums comparable to that defined in 2.4.1.9.1, and may be performed in any order. Any or all of the following UCR bits may be set by the execution of this instruction; Exponent Overflow, Exponent Underflow, Floating Point Loss of Significance and Floating Point Indefinite. When any of these condition bits are set, the final sum is undefined.

The number of intermediate sums formed in the execution of this instruction is of interest because multiple floating point add operations are sensitive to the order in which the adds are performed for certain operands. The order of the summation instruction is model-dependent. However, each processor model shall always form the sum in an identical order for every execution with identical input fields. Thus, each processor model shall always produce identical results for identical input fields.

Because each processor model may implement a different order in forming the final sum, the result of this instruction with identical input fields may vary (within the constraints of the rules of floating point arithmetic) from one model processor to another. A condition bit or bits may be set on one model but not on another when executing this instruction with identical input fields.

2.12.9 Merge Vector

Merge vector, V(Ak) partially replaced by V(Aj) per mask V(Ai)

54jkiD (Ref. 191)

This instruction shall replace the first element of V(Ak) with the first element of V(Aj) if bit 0 is set in the first element of V(Ai). If bit 0 is clear, the first element of V(Ak) shall be left unchanged. This operation is repeated for successive elements until the required number of operations has been performed.

2.12.10 Gather/Scatter Vectors

a. Gather vector, V(Ak) replaced by gathered V(Aj) with interval Xi

55jkiD (Ref. 192)

b. Scatter vector, V(Ak) replaced by scattered V(Aj) with interval Xi

56jkiD (Ref. 193)

Designator i designates register Xi which contains the interval for the gather or scatter instruction. This interval may be either positive (including zero) or negative.

The execution of the gather and scatter instructions shall be undefined with respect to the generated results for every case in which the source and destination fields overlap. (Coincidence in the leftmost and rightmost positions does not cause the instruction to be defined as for other vector instructions.)

The processor need not prevalidate all of the PVAs generated by the gather or scatter instructions for Address Specification Error or Page Table Search Without Find before beginning to store the results into central memory. These two exceptions may be tested as the addresses are generated, and if either is detected, the following shall occur:

- The storing of results into memory shall halt either with or before the result associated with the exception. (Note that some results may be stored prior to instruction halt.)
- The address which could not be translated into a real memory address shall be placed into the UTP register.
- The appropriate bit in the MCR shall be set and the action specified by table 2.8-1 taken.
- The PVA in P at the time of the interrupt shall point to the gather or scatter instruction which attempted the central memory reference which resulted in the interrupt.

Upon returning to the process containing the gather or scatter instruction, the entire instruction shall be reinitiated.

If the interval contained in Xi, when repetitively added to the contents of the appropriate A register, causes the byte number portion of the address to exceed $2^{31}-1$ or to become negative (both cases set bit 32), an Address Specification Error shall be recorded and the instruction halted as described above.

Gather Instruction

This instruction obtains the first element from V(Aj) and stores it as the first element of V(Ak). The second element to be stored in V(Ak) is taken from the address formed by adding the rightmost 32 bits of Xi, shifted left three places with zero fill, to the rightmost 32 bits of Aj. Successive elements in V(Ak) are taken from the address formed by adding the rightmost 32 bits of Xi, shifted left three places with zero fill, to the rightmost 32 bits of the previous address. The n^{th} (1,2,3...n,...) element of V(Ak) is replaced by V(Aj) whose address is (Aj)+8*(n−1)(Xi). The contents of register Xi are not altered by the execution.

Thus, contiguous vector V(Ak) is formed by gathering elements from V(Aj) at interval Xi.

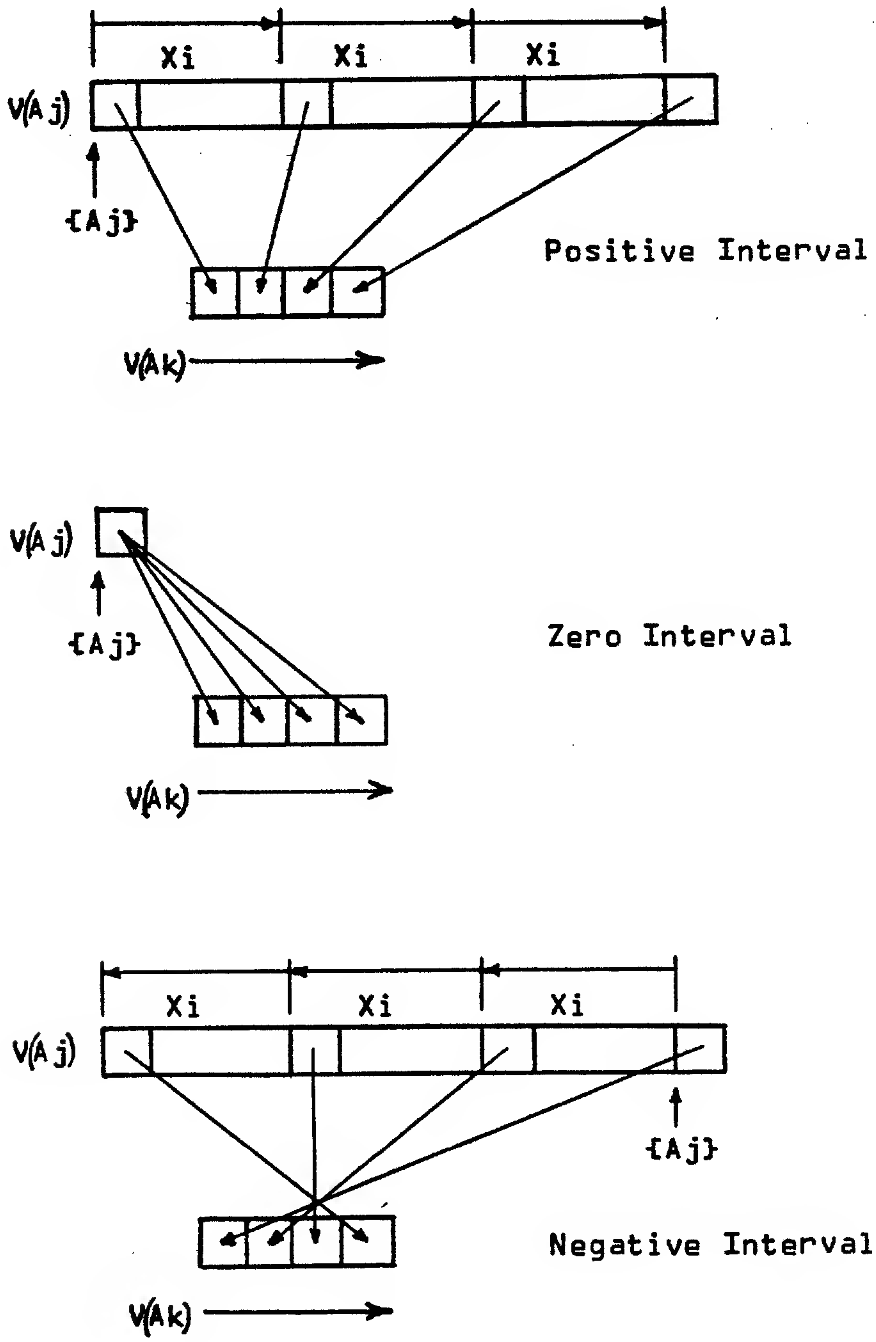


Figure 2.12-1. Gather Instruction

Scatter Instruction

This instruction obtains the first element from V(Aj) and stores it as the first element of V(Ak). The second contiguous element from V(Aj) is stored into V(Ak) at the address formed by adding the rightmost 32 bits of Xi, shifted left three places with zero fill, to the rightmost 32 bits of Ak. Successive elements from V(Aj) are stored into the addresses formed by adding the rightmost 32 bits of Xi, shifted left three places with zero fill, to the rightmost 32 bits of the previous address. The n^{th} (1,2,3,...n,...) element of V(Aj) is stored into V(Ak) at (Ak)+8*(n−1)(Xi).

Thus, the contiguous elements from V(Aj) are scattered in V(Ak) at interval Xi.

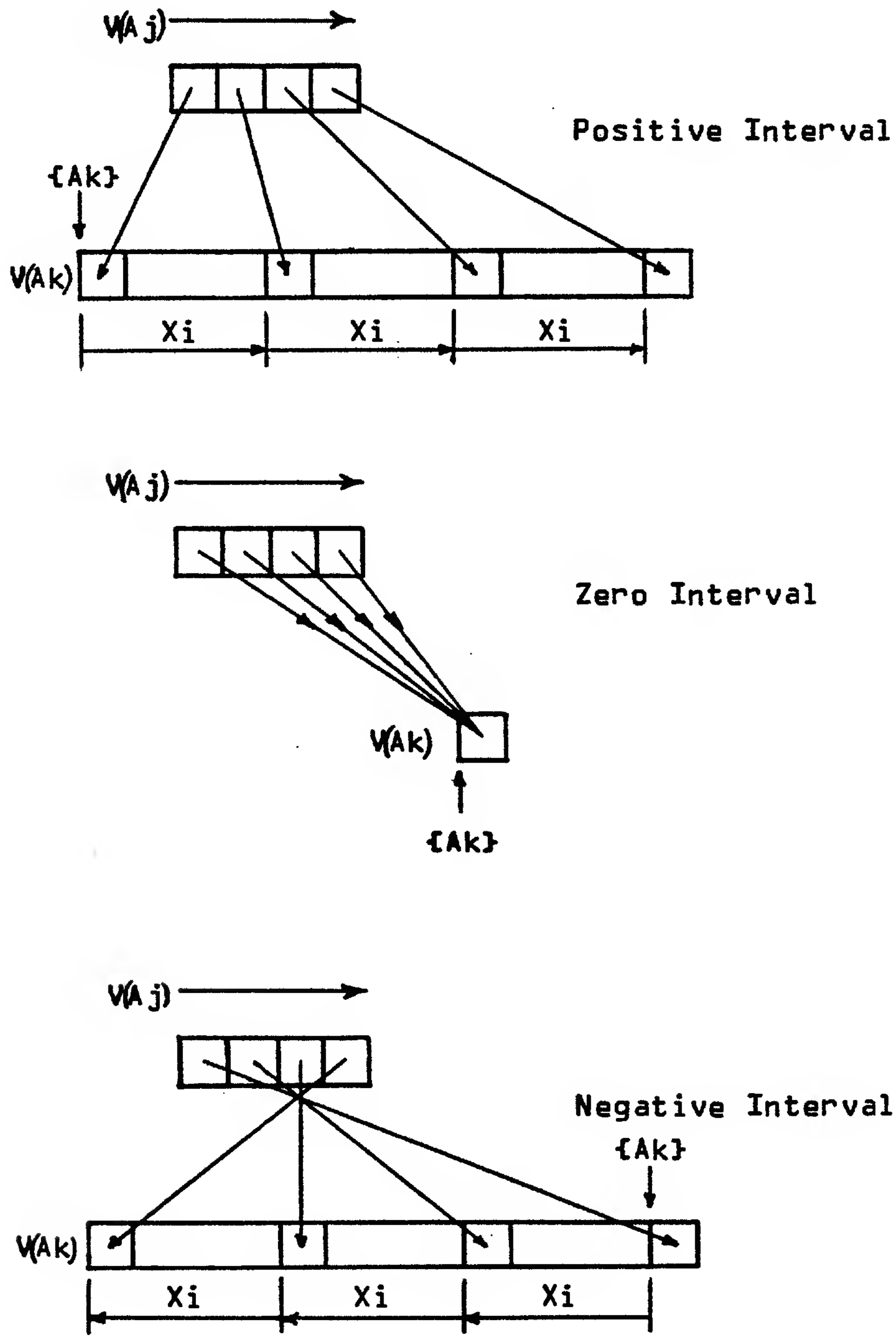


Figure 2.12-2. Scatter Instruction

2.12.11 Floating Point Vectors - Triad

a. Floating point vector triad, V(Ak) replaced by [V(Ai) times (X0)] plus V(Aj)

58jkiD (Ref. 195)

b. Floating point vector triad, V(Ak) replaced by [V(Ai) times (X0)] minus V(Aj)

59jkiD (Ref. 196)

c. Floating point vector triad, V(Ak) replaced by [V(Aj) plus V(Ai)] times (X0)

5AjkiD (Ref. 197)

d. Floating point vector triad, V(Ak) replaced by [V(Aj) minus V(Ai)] times (X0)

5BjkiD (Ref. 198)

These instructions shall perform the indicated floating point arithmetic operations on the first element from V(Aj) and V(Ai) and the floating point constant contained in X0 and store the result as the first element of V(Ak). The arithmetic operation inside the brackets is done first. The operation is repeated for successive elements until the required number of operations has been performed. The contents of register X0 are not altered by the execution of these instructions.

2.12.12 Floating Point Vector Dot Product

Floating point vector dot product, Xk replaced by summation of [V(Aj) times V(Ai)]

5CjkiD (Ref. 199)

This instruction shall perform a product on the first element from V(Aj) and from V(Ai). This product is repeated for successive elements until the required number of operations has been performed. This instruction shall add together all of the products and store that sum in Xk. The individual add operations which together form this instruction are single precision sums comparable to that defined in 2.4.1.9.1, and may be performed in any order. Any or all of the following UCR bits may be set by the execution of this instruction: Exponent Overflow, Exponent Underflow, Floating Point Loss of Significance, and Floating Point Indefinite. When any of these condition bits are set, the final sum is undefined. The number and order of intermediate sums is as defined in 2.12.8.

Due to the order of the summation operation, the result given by this instruction on two different mainframe models may differ in the rightmost bit positions even if the source operands are identical. However, the same mainframe models shall yield the same result for each execution of this instruction with identical operands.

2.12.13 Gather/Scatter Vectors - Index List

a. Gather vector, V(Ak) replaced by gathered V(Aj) per index list V(Ai)

5DjkiD (Ref. 200)

b. Scatter vector, V(Ak) replaced by scattered V(Aj) per index list V(Ai)

5EjkiD (Ref. 201)

The index list from V(Ai) contains a list of indexes for the gather or scatter instruction. These indexes may be positive (including zero) or negative. If the indexes from V(Ai), when added to the contents of the appropriate A register, causes the byte number portion of the address to exceed $2^{31}-1$ or to become negative (both cases set bit 32), an Address Specification Error shall be recorded and the instruction halted. The field overlap and prevalidation for these instructions is comparable to that defined in 2.12.10.

Gather Instruction - Index List

This instruction obtains the first element from V(Ai). The first element to be stored in V(Ak) is taken from the address formed by adding the rightmost 32 bits of the first element from V(Ai), shifted left three places with zero fill, to the rightmost 32 bits of (Aj). Successive elements in V(Ak) are taken from the addresses formed by adding the rightmost 32 bits of the successive elements from V(Ai), shifted left three places with zero fill, to the rightmost 32 bits of (Aj). The n^{th} (1, 2, 3, ...n, ...) element of V(Ak) is replaced by the element of V(Aj) whose address is (Aj) + 8 times the n^{th} element of V(Ai).

Thus, contiguous vector V(Ak) is formed by gathering elements from V(Aj) at indexes from list V(Ai).

Scatter Instruction - Index List

This instruction obtains the first elements from V(Ai) and V(Aj). The first element from V(Aj) is stored into V(Ak) at the address formed by adding the rightmost 32 bits of the first element from V(Ai), shifted left three places with zero fill, to the rightmost 32 bits of (Ak). Successive elements in V(Aj) are stored into V(Ak) at the address formed by adding the rightmost 32 bits of the successive elements from V(Ai), shifted left three places with zero fill, to the rightmost 32 bits of (Ak). The n^{th} (1, 2, 3, ...n, ...) element of V(Aj) is stored into V(Ak) at (Ak) + 8 times the n^{th} element of V(Ai).

Thus, contiguous elements from V(Aj) are scattered in V(Ak) at indexes from list V(Ai).

3.0 VIRTUAL MEMORY MECHANISM

3.1 GENERAL DESCRIPTION

Central memory shall be addressed by means of virtual memory addresses. This section concerns itself with the definition, formation and translation of virtual memory addresses as well as the access protection mechanisms provided in systems.

3.1.1 Level of Addresses

Within systems, three levels of central memory addresses shall be recognized: process virtual address (PVA), system virtual address (SVA), and real memory address (RMA).

Each process virtual address (PVA) shall consist of three major components: A segment number (SEG), a byte number (BN) and a ring number (RN). The process virtual address shall be local to a process and shall be translated into a global, system virtual address (SVA) by means of the process segment table. The translation process shall consist of converting the process segment number (SEG) into the system's active segment identifier (ASID) and checking the appropriate access controls to the segment.

To address central memory, the system virtual address (SVA) shall be further translated into the real memory address (RMA) through the system page table. Each paged segment shall be divided into pages and shall be allocated into real memory accordingly.

3.1.2 Address Components

The process virtual address (PVA) shall consist of a segment number (SEG), a byte number (BN) and a ring number (RN). The RN shall be used for access control and the combination of the SEG and BN shall specify a byte address.

The system virtual address (SVA) shall consist of an active segment identifier (ASID) and a byte number (BN). Within the SVA, the BN shall be further divided into subfields. The BN shall consist on a page number (PN) and a page offset (PO).

The concepts of segment and page are discussed in the following subparagraphs.

3.1.2.1 Segments

In systems, data and programs shall be organized into units consisting of segments. Each segment shall be defined to be a contiguous bit-string of information with a maximum length less than or equal to 2^{31} bytes. An instruction (or datum) shall be identified (addressed) by the segment name to which it belongs and the byte name within the segment where it is located. The segment shall be defined to be the basic unit of information sharing among different processes. In order to retain a level of flexibility in naming, each process shall identify a segment with its own (process) segment number. The 12-bit process segment number shall be translated into a 16-bit system (global) segment identifier, called the active segment identifier (ASID), by means of the process segment table. The process segment table shall effectively define the process virtual addressing space. The 12-bit process segment name shall limit the maximum number of addressable segments by a process to 4096.

The 16-bit active segment identifier (ASID) shall consist of a segment name used by the system to identify each segment currently active in the system. To each active segment, one, and only one ASID shall be assigned even though it might correspond to more than one process segment number. From the perspective of the system software, the ASID shall provide a "short" name for the more permanent segment (file) name used in the information storage subsystems. The translation from the permanent name to the "short" ASID shall be accomplished by the software.

All active pages within a given segment must exist in the same memory, either local or shared (see 4.1.6). Duplicate copies of the same page cannot coexist anywhere in memory.

3.1.2.2 Pages

To facilitate mapping segments into real memory, and to enable management of very large central memories, the segments shall be subdivided into pages. Page sizes shall vary between a minimum of 512 bytes and a maximum of 64K bytes. In any given processor, the page size shall be fixed. Within each page, addressing shall be performed to the byte. The total hierarchy then, shall be:

```
--------------
\  SEGMENT   /
 \----------/
  \  PAGE  /
   \------/
    \BYTE/
     \  /
      \/
```

Figure 3.1-1. Address Component Hierarchy

It must be noted, however, that in general, users shall refer only to a segment and a byte number within a segment. Pages shall be transparent to the user in much the same way that central memory banks shall be transparent to users in real memory.

3.1.3 Real Memory Address

The real memory address (RMA) shall be defined as a 32-bit byte address with the leftmost position referred to as the sign bit:

32	33 — 63
S	(31)

For certain configurations, bit 33 is used to select the central memory port in accordance with paragraph 2.10.1.1. RMA bits 34 and 35 are reserved. The actual central memory size shall be a system installation parameter.

3.2 PROCESS VIRTUAL ADDRESS

The following paragraphs define the format of the process virtual address and the logical algorithms used for translating the process virtual address into the system virtual address.

3.2.1 Format

The process virtual address (PVA) shall constitute the effective address presented by a program (process) to address the central memory. The formation of the PVA shall be determined by the instruction repertoire and the manner in which the various fields from each instruction shall be used to form the effective address. The format of the PVA shall be as follows:

0	16	20	32	— 63
////////////	RN	SEG	S	
			BN	BN

3.2.1.1 Ring Number

The ring number (RN) shall consist of a four bit field contained in bit positions 16 through 19 of each PVA. It shall be used for access validation as discussed in section 3.6.

RN shall also be used as a special flag such that a ring number of zero, (RN=0) shall denote an unlinked Pointer. (See 2.8.1.13.)

The test for ring number equal zero (RN=0) shall be performed at, and only at, the following points:

Instruction	Object tested for Ring No. ≠ Zero	Action taken when Ring No. = Zero
Return (Op. 04)	Any A register as read from the Previous Stack Frame Save Area (2.6.1.4)	Sets MCR60 at the completion of instruction execution. Does not inhibit instruction execution. Does not alter the UTP.
Pop (Op. 06)	Register A1 or A2 as read from the Previous Stack Frame Save Area (2.6.1.5)	
Load Multiple (Op. 80) Load Address (Op. 84) Load Address, Indexed (Op. A0)	The quantity as read from memory that is to be loaded into an A register (2.2.1.6 and 2.2.1.7)	

[See section 1.6 for a list of the C180 models that do not perform the RN=0 test on these five instructions.]

Thus, at all other points, RN=0 shall be interpreted as the highest privileged ring number and subjected only to the tests as specified in 3.6.2.1 and 3.6.2.2.

3.2.1.2 Segment Number

The segment number (SEG) shall consist of a 12-bit field contained in bit positions 20 through 31 of each PVA. It shall be used to identify a single segment from all other segments addressable by the process.

3.2.1.3 Byte Number

The byte number (BN) shall consist of a 32-bit field contained in bit positions 32 through 63 of each PVA. It shall specify the byte location (or displacement) within a segment. Bit position 32 of each PVA shall constitute the sign bit of the BN field and must be in the zero state. In the one (negative) state, this bit shall generate an Address Specification interrupt at the time it is used to address central memory.

3.3 PROCESS SEGMENT TABLE

The process virtual address shall be translated into the system virtual address by means of the process segment table. The process segment table shall be specified by two values: the segment table address (STA) and the segment table length (STL). The STA shall represent the real address of the first entry of the process segment table. Each entry within a segment table shall be 64 bits long and shall be accessed by indexing the STA with the appropriate segment number. The STL, plus one, shall represent the number of usable entries in the associated segment table. The segment number, (which is applied as an index to the STA) must be less than or equal to the value of the STL. The process segment table shall effectively define the process virtual address space. The maximum number of entries which may be contained in a segment table shall be 4096.

The hardware implementations may make the following assumptions:

- The processor may assume that valid entries in the Process Segment Table at the beginning of an instruction execution will be present during the remainder of the execution of that same instruction with no change.
- The processor may assume that any change to a valid or invalid entry in a Process Segment Table which is in use or has been used will be appropriately accompanied by a MAP purge operation. This is to support the hardware implementations which use a change in the job state STA to purge the MAP of entries based on segment table entries. Thus, a job process which exchanges to C180 monitor, which in turn alters the job segment table and then returns to the same job requires a software initiated purge of the altered segment table entries from the MAP.
- The processor may assume that any change to the Segment Table Length will be accompanied by a purge of those valid and/or invalid entries which are affected by the Length change.

3.3.1 Segment Descriptors

Each of the 64-bit entries contained in the segment table shall be referred to as segment descriptors and shall be formatted as follows:

0	2	4	6	8	12	16 ... 31
VL	XP	RP	WP	R1	R2	ASID

32	34	40 ... 63
01	Lock	//

3.3.1.1 Control Fields

The eight control bits contained in each segment descriptor, (bit positions 00 through 07), shall be grouped into four 2-bit fields referred to as VL, XP, RP, and WP. Each of these four groups shall be decoded and translated as follows:

VL		RP	
00	Invalid Entry	00	Nonreadable Segment
01	(Reserved)	01	Read Controlled by Key/Lock
10	Regular Segment	10	Read Not Controlled by Key/Lock
11	Cache By-Pass Segment	11	Binding Section Segment-Read not Controlled by Key/Lock

XP		WP	
00	Nonexecutable Segment	00	Nonwritable Segment
01	Nonprivileged Executable Segment	01	Write Controlled by Key/Lock
10	Local Privileged Executable Segment	10	Write not Controlled by Key/Lock
11	Global Privileged Executable Segment	11	(Reserved)

Notes: Binding Section Segments shall be created by the System software (linker) and shall be used during the execution of Call instructions as described in section 2.6 of this specification. Segments having RP=11 may be read as if RP=10.

Read, Write and Execution privileges are described in section 3.6 of this specification.

The processor shall never use data from the cache when reading from a cache bypass segment. It is permissible but not required to update the cache by bringing the data into cache; however, if the processor brings the data into cache it must also purge cache on writes to a cache bypass segment. (This does not restrict the processor's freedom to purge entries from cache at any time.) The engineering specifications shall describe the processor operation relative to cache bypass segments.

3.3.1.2 Access Validation Fields

The R1 and R2 fields shall all consist of four bits each. Bit 32 shall be set to zero and bit 33 shall be set to one (as described in section 3.6.3.1). Bits 34 through 39 form the lock field. These fields shall constitute inputs to the access control mechanism in order to perform access validation as described in section 3.6 of this specification.

3.3.1.3 Active Segment Identifier

The active segment identifier (ASID) shall consist of a 16-bit field and shall constitute a global name which identifies a single segment from all other segments currently active in the system.

3.3.1.4 Conversion to System Virtual Address

The process segment table entries shall be used primarily to validate central memory accesses. However, they shall also be utilized to convert the PVA to a system virtual address (SVA), by substituting a 16-bit active segment identifier for the 12-bit process segment number. The formation of the SVA is illustrated in figure 3.3-1.

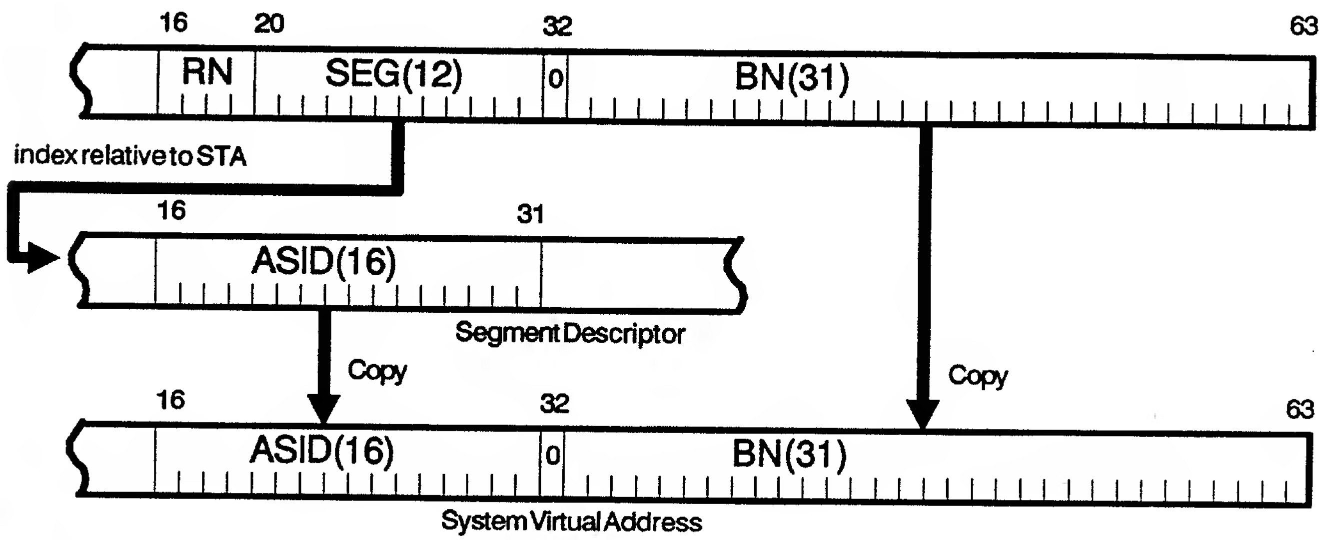


Figure 3.3-1. Conversion of PVA to SVA

3.4 SYSTEM VIRTUAL ADDRESS

This section specifies the format of the system virtual address and the logical algorithms used for translating the system virtual address into the real memory address. The system virtual address (SVA) shall represent a global address shared by all processes active in the system. An SVA shall consist of an active segment identifier (ASID) and a byte number (BN). The format of an SVA shall be defined as follows:

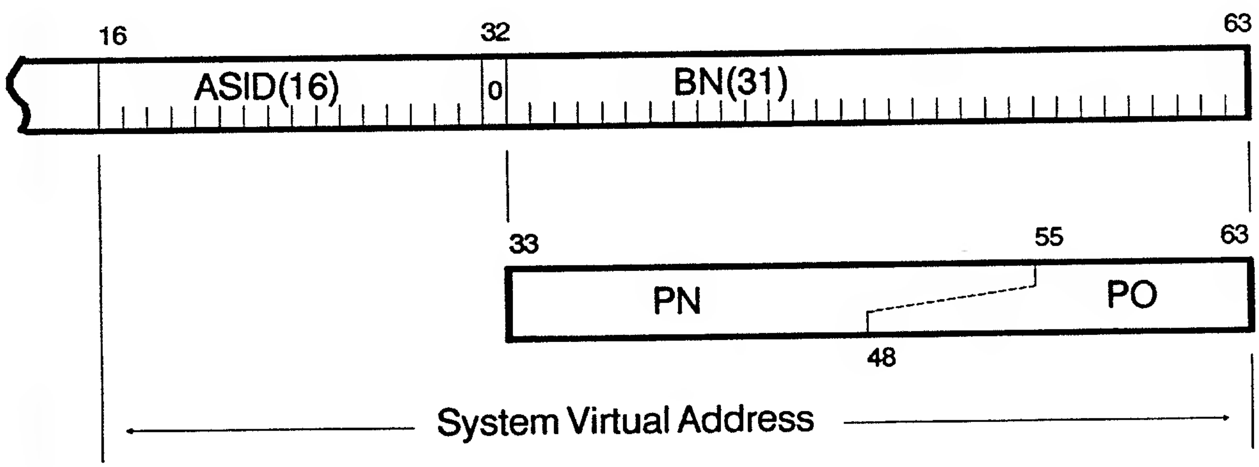


Figure 3.4-1. System Virtual Address (SVA)

3.4.1 Active Segment Identifiers

The active segment identifier (ASID) shall consist of a 16-bit field contained in bit positions 16 through 31 of the SVA. The ASID shall represent a global name that identifies the segment from all other currently active segments in the system. Two processes which are sharing a segment may have different (process) segment numbers (SEG) to address that segment, but must have the same ASID.

3.4.2 Byte Number

The byte number (BN) shall consist of a 31-bit field contained in bit positions 33 through 63 of the SVA. It shall specify the byte location (or displacement) within a segment.

Within the BN, the address translation mechanism shall further recognize two subfields: a page number (PN) and a page offset (PO).

Note: It must be stressed that these subfields are recognized only by the address translation mechanism and are transparent to general programs.

3.4.2.1 Page Number

The page number (PN) field shall be variable in size and range from 15 to 22 bits, as determined by the page size of the system. The page size shall be fixed on a per installation basis and shall not vary while the system is running. The actual size of the page number field shall be contained as a mask in the page size mask register.

3.4.2.2 Page Size Mask Register

The page size mask (PSM) register shall be set such that its use against bits 48 through 54 of the SVA shall allow the separation of the page number from the page offset. Bit positions 33 through 47 of the SVA shall be automatically included in the page number, and bits 55 through 63 shall be automatically included in the page offset. The page size mask shall consist of 7 bits and shall represent a logical prefix vector with (7-U) ones, followed by U zeros, where the page size is 2^0 x 512 bytes. For example, U=2 yields a page size of $2^{(2+9)}$=2048 bytes. The corresponding page size mask would be set to: "1111100."

The software is responsible to always clear the rightmost two bits of the PSM. The hardware may ignore these two bits. This results in six page sizes as follows:

PSM	Page Size
111 1100	2K Bytes
111 1000	4K Bytes
111 0000	8K Bytes
110 0000	16K Bytes
100 0000	32K Bytes
000 0000	64K Bytes

3.4.2.3 Page Offset

The page offset (PO) shall represent the displacement of the central memory location to be accessed relative to the page boundary. This field shall vary with the page size and range from 9 to 16 bits.

The formation of the page number and the page offset from the byte number and the page size mask is illustrated by figure 3.4-2 as follows:

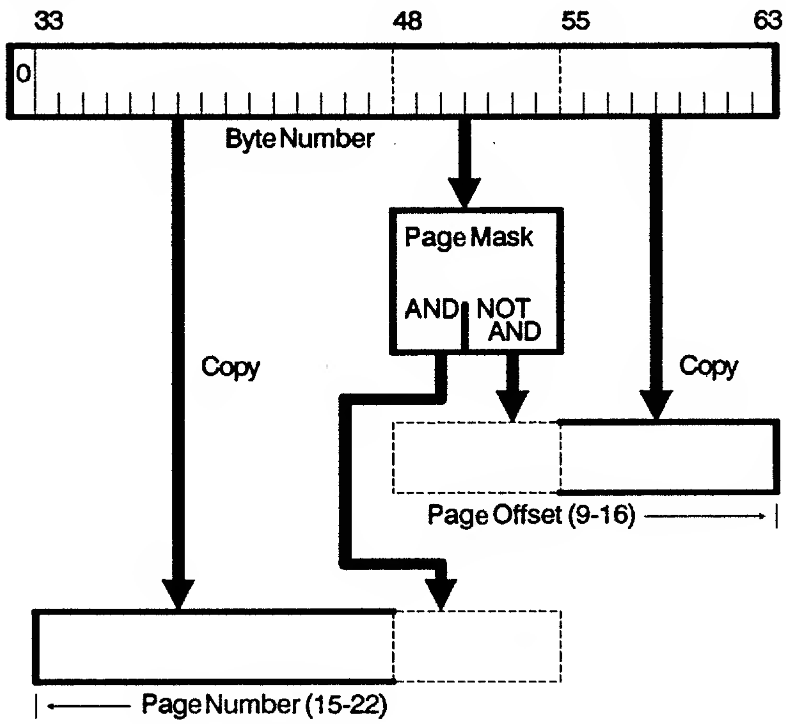


Figure 3.4-2. Formation of Page Number and Page Offset

3.5 SYSTEM PAGE TABLE (SPT)

System virtual addresses (SVAs) shall be translated into real memory addresses (RMAs) by means of the system page table. Each page currently allocated in the central memory shall have a corresponding entry (page descriptor) in the system page table which contains the ASID, the page number, and the corresponding physical address where the page starts in real memory.

The system page table shall be specified by two values: the page table address (PTA) (2.5.1.3) and the page table length (PTL) (2.5.1.4). The page table address shall represent the real address of the first entry of the system page table which must be 0, modulo page table length. Each page table entry shall consist of a 64-bit word or page descriptor (3.5.1).

The page descriptor required to translate an SVA into an RMA shall be found by first forming an index (3.5.2.1) into the system page table and then by a search (3.5.2.2) of up to 32 entries to find the descriptor corresponding to the SVA to be translated.

There are two versions of the Page Table Length (PTL) register:

8-bit PTL

The page table length shall consist of 8 bits and shall specify the length as 2^n x 4096 bytes for n = 0,1,2...8 (see 2.5.1.4) as shown in tables 3.5-1 and 3.5-2.

14-bit PTL

The page table length shall consist of 14 bits and shall specify the length as 2^n x 4096 bytes for n = 0,1,2...14 (see 2.5.1.4) as shown in tables 3.5-1 and 3.5-2.

Tables 3.5-1 and 3.5-2 contain a number of entries for each of the valid PTL contents in the lefthand portion of the table. The righthand portion of each table contains the maximum size central memory for the specific PTL value and each of the legal page sizes. Table 3.5-1 contains the maximum theoretical central memory size possible with a 100% full page table. However, since a practical arrangement of the entries in a page table requires a density of about 25%, Table 3.5-2 shows the expected correlation of PTL and central memory size for a more practical page table.

Note: In tables 3.5-1 and 3.5-2, memory sizes approximate the actual value, but are stated in powers of 2 (decimal) for the convenience of the reader.

	PAGE TABLE LENGTH		PAGE SIZE					
	PTL	Number of Entries	2K	4K	8K	16K	32K	64K
	00	512	1M	2M	4M	8M	16M	32M
	01	1K	2M	4M	8M	16M	32M	64M
	03	2K	4M	8M	16M	32M	64M	128M
	07	4K	8M	16M	32M	64M	128M	256M
	0F	8K	16M	32M	64M	128M	256M	512M
	1F	16K	32M	64M	128M	256M	512M	1G
8-bit	3F	32K	64M	128M	256M	512M	1G	2G
PTL	7F	64K	128M	256M	512M	1G	2G	4G
limit	FF	128K	256M	512M	1G	2G	4G	8G
	01FF	256K	512M	1G	2G	4G	8G	16G
	03FF	512K	1G	2G	4G	8G	16G	32G
	07FF	1M	2G	4G	8G	16G	32G	64G
	0FFF	2M	4G	8G	16G	32G	64G	128G
	1FFF	4M	8G	16G	32G	64G	128G	256G
	3FFF	8M	16G	32G	64G	128G	256G	512G

Table 3.5-1. Theoretical Maximum Central Memory (100% full Page Table)

	PAGE TABLE LENGTH		PAGE SIZE					
	PTL	Number of Entries	2K	4K	8K	16K	32K	64K
	00	512	256K	512K	1M	2M	4M	8M
	01	1K	512K	1M	2M	4M	8M	16M
	03	2K	1M	2M	4M	8M	16M	32M
	07	4K	2M	4M	8M	16M	32M	64M
	0F	8K	4M	8M	16M	32M	64M	128M
	1F	16K	8M	16M	32M	64M	128M	256M
8-bit	3F	32K	16M	32M	64M	128M	256M	512M
PTL	7F	64K	32M	64M	128M	256M	512M	1G
limit	FF	128K	64M	128M	256M	512M	1G	2G
	01FF	256K	128M	256M	512M	1G	2G	4G
	03FF	512K	256M	512M	1G	2G	4G	8G
	07FF	1M	512M	1G	2G	4G	8G	16G
	0FFF	2M	1G	2G	4G	8G	16G	32G
	1FFF	4M	2G	4G	8G	16G	32G	64G
	3FFF	8M	4G	8G	16G	32G	64G	128G

Table 3.5-2. Practical Maximum Central Memory (25% full Page Table)

3.5.1 Page Descriptors

System page table entries shall consist of 64 bits each, and shall be referred to as page descriptors. Each page descriptor shall identify a page frame to be accessed as well as record usage of that page frame. Page descriptors shall be formatted as follows:

0 1	2 3	4 41	42 63
V,C	U,M	SEGMENT/PAGE IDENTIFIER (38)	PAGE FRAME ADDRESS (22)

3.5.1.1 Control and Status Fields

The four control bits in positions 00 through 03 are the Valid (V), Continue (C), Used (U), and Modified (M) bits. These shall be decoded and translated as follows:

Bit No.	Set	Clear	
00	Valid entry	Invalid entry	Control
01	Continue search	May stop search	
02	Used	Unused	Status
03	Modified	Unmodified	

The Valid and Continue bits shall provide the means for controlling the search of the page table for the proper SVA (see 3.5.2.2).

The hardware shall set the Used bit when the page table entry is used for address translation. The hardware never clears this bit.

The hardware shall set the Modified bit when the page table entry has been used for address translation which will result in a write into the associated page. The hardware never clears this bit.

3.5.1.2 Segment/Page Identifier (SPID)

The 38-bit Segment/Page Identifier field shall identify the System Virtual Address for the Page Descriptor Entry. It shall include the 16-bit ASID and the 22-bit Page Number. For a system in which the page size is larger than 512 bytes, i.e., Page Number less than 22 bits, zeros shall be correspondingly added in the rightmost bit positions. (Nonzero bits cause the operation of the address translation hardware to be undefined.)

3.5.1.3 Page Frame Address

This 22-bit field is the physical address of the Page Frame.

When the page size is larger than 512 bytes, zeros must be present in the rightmost bit positions to obtain proper alignment. (Nonzero bits cause the operation of the address translation to be undefined.)

3.5.2 Allocation of Page Descriptors

3.5.2.1 Location of a Page Descriptor in the Page Table

As shown in figure 3.5-1, the page table descriptor required to translate an SVA into an RMA shall be found by first forming an index into the system page table and then by a search (3.5.2.2) of up to 32 entries in order to find the descriptor corresponding to the SVA to be translated.

The index into the page table is a pseudo random mapping (hashing) of a large (38-bit) address space into a smaller (16 or less bit) address space. Because it is a many-to-one mapping, the SVA's associated with several pages may map into the same index into the page table.

These multiple entries will be placed in the page table after (higher addresses) the entry indicated by the index, thus, potentially requiring a search of additional entries as described in 3.5.2.2.

8-bit PTL

The address of the first entry searched in the page table shall be formed from the SVA as described below (see figures 3.5-1 and 3.5-2).

1. The EXCLUSIVE OR of the 16-bit ASID and the rightmost 16 bits of the Page Number shall be formed.
2. A logical AND shall be performed with the leftmost 8 bits of the 16 bits resulting from step 1 and the page table length. This reduces the index to a value within the current page table length.
3. The 16-bit quantity from step 2 shall have four zero bits catenated on the right to form the index into the page table.
4. This index is merged with the page table address which is 0, modulo the page table length, thus, having zeros where the index is inserted (2.1.1.3).

14-bit PTL

The address of the first entry searched in the page table shall be formed from the SVA as described below (see figures 3.5-1 and 3.5-3).

1. The EXCLUSIVE OR of the 16-bit ASID and the rightmost 16 bits of the Page Number shall be formed and shall be the rightmost 16 bits of the Hash Index. The EXCLUSIVE OR of the leftmost 6 bits of the ASID and the rightmost 6 bits of the Page Number shall be formed and shall be the leftmost 6 bits of the 22-bit Hash Index.
2. A logical AND shall be performed with the leftmost 14 bits of the 22-bit Hash Index and the 14-bit PTL register. This reduces the index to a value within the current page table length.
3. The 22-bit quantity from step 2 shall have four zero bits catenated on the right to form the index into the page table.
4. This index is merged with the page table address which is 0, modulo the page table length, thus having zeros where the index is inserted (2.1.1.3).

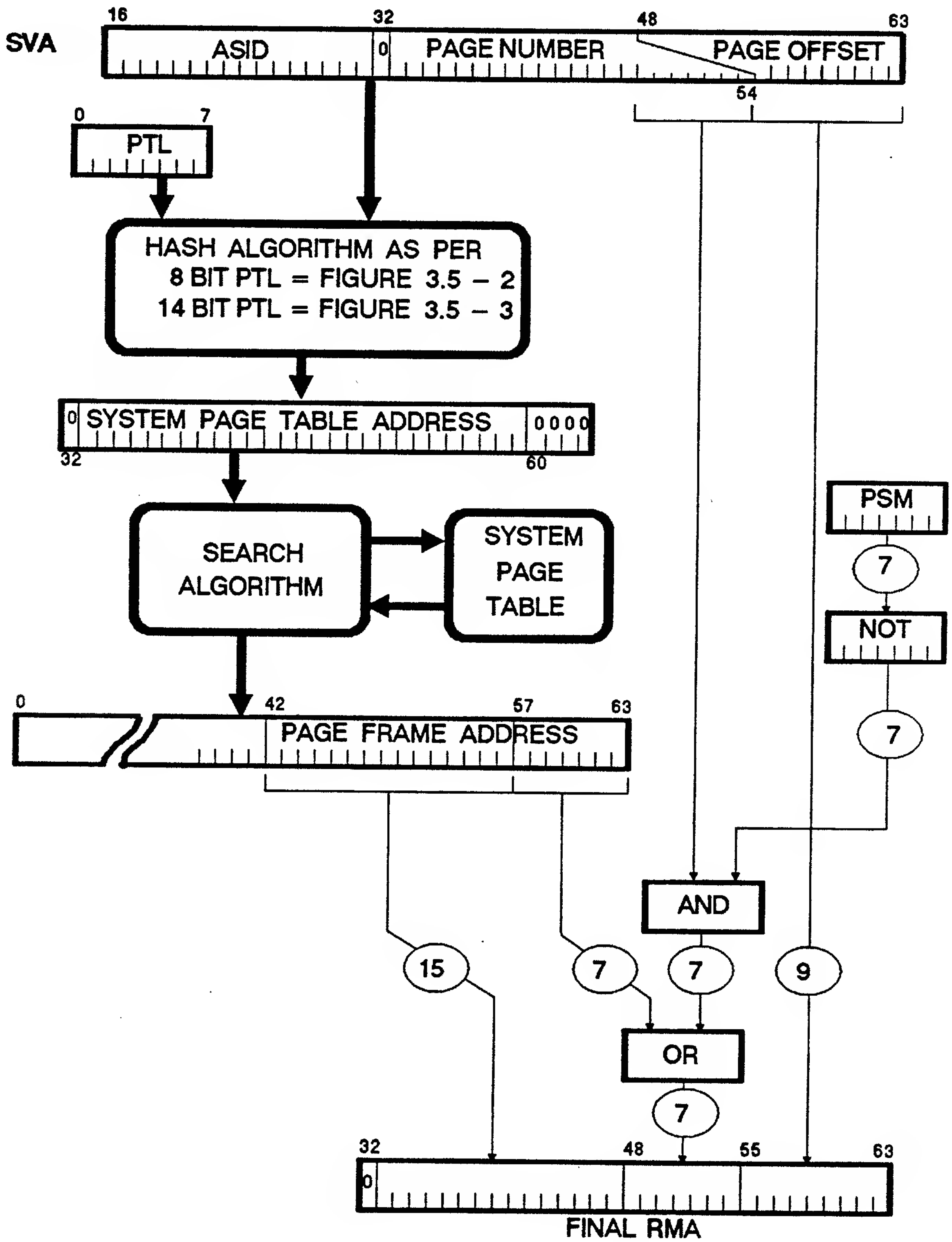


Figure 3.5-1. Transformation of SVA to RMA

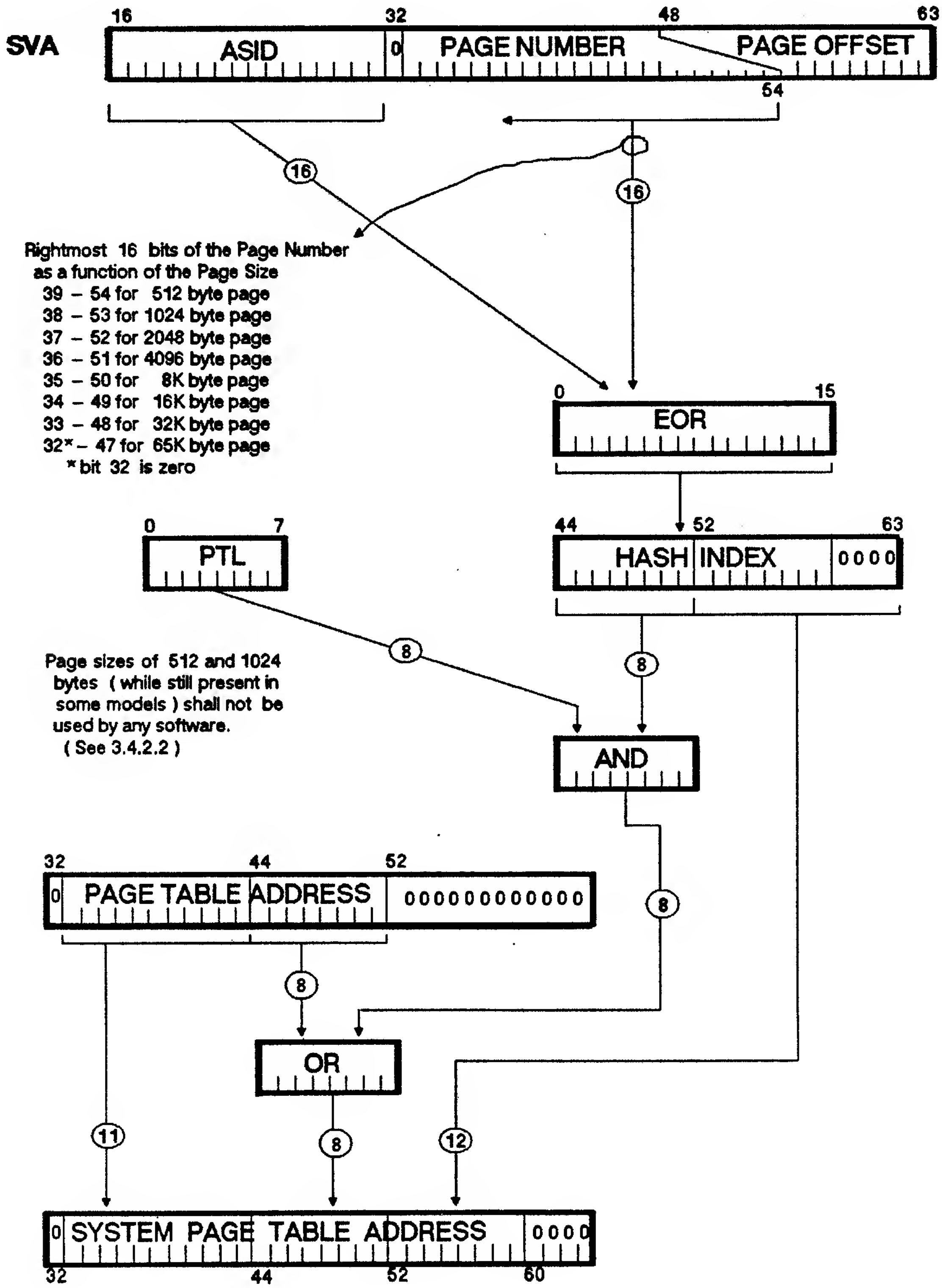


Figure 3.5-2. Eight-bit Page Table Length

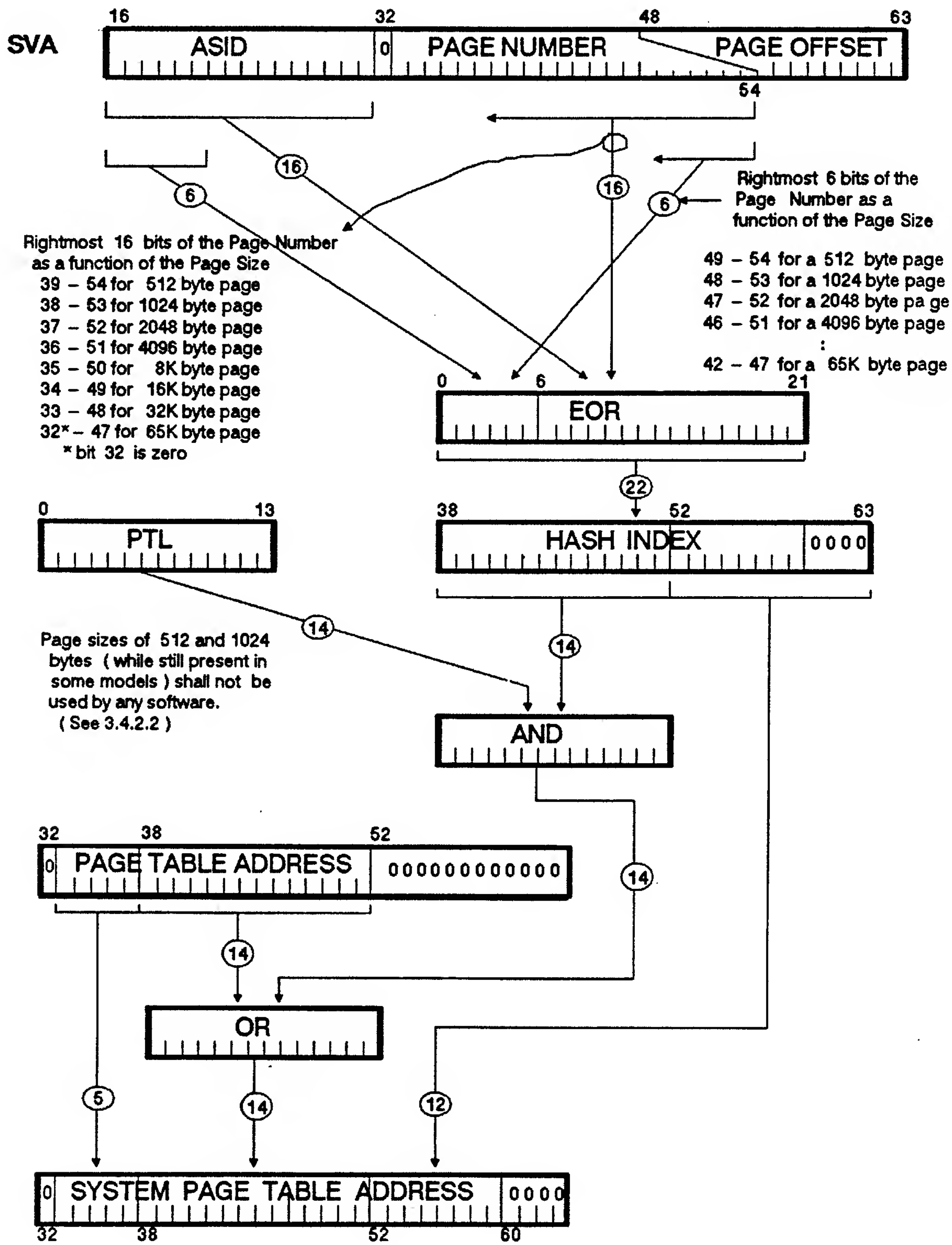


Figure 3.5-3. Fourteen-bit Page Table Length

3.5.2.2 Search for Page Descriptor in the Page Table

The processor shall test up to 32 entries in the system page table while searching for an entry whose Segment/Page Identifier matches that of the SVA initiating the search. The 32 entries tested are those contiguous, valid or invalid, entries which start at the location in the system page table indicated by the hash index described in the previous paragraph and include the next 31 higher address words. A search which encounters the end of the page table before searching 32 entries shall continue the search at the first entry in the page table.

In general, the processor conducts the search in sequential order by increasing address. For optimal performance, therefore, the operating system should put entries into the page table in this same order. However, the processor may alter the sequence of the search under certain circumstances. In these cases, the search shall still be complete and accurate but may result in more entries being tested before finding the match than would have been tested on a strictly sequential search. The processor may, but need not, test any entries after the first sequential entry having the Continue flag clear. The interpretation of this Continue bit is independent of the Valid bit. The processor shall never alter the state of the Valid bit or the Continue bit as part of searching the page table.

The Valid bit, when set, indicates to the processor that this entry should be tested for a possible match to the current SVA. The processor shall always test the validity of any Page Descriptor during the first (and perhaps only) utilization of that Page Descriptor during an instruction execution (see 2.6.2.1). When a Page Descriptor is used more than once during the same instruction, the processor shall either ignore the Valid bit after once validating an entry or may continue to test validity as long as any subsequent Page Fault resulting from an asynchronously cleared Valid bit results in inhibiting the instruction execution.

The hardware implementations are free to make the following assumptions:

- The processor may assume on any search of the system page table that no more than one entry (valid or invalid) exists within this set of 32 entries which has a Segment/Page Identifier satisfying the search.
- The processor may assume during the execution of any given instruction that no other processor changes the Continue bit in any entry involved between the hash index and the required Page Descriptor in any search associated with this execution; however, another processor may change the Valid bit at any time. Note that the Continue bit in the required descriptor may be changed at any time.
- The processor may assume that valid entries in the system page table at the beginning of an instruction execution will be present during the remainder of the execution of that same instruction, although the Valid bit may have been cleared during execution.

3.5.2.3 Formation of the Real Memory Address (RMA)

The logical algorithm for translating a system virtual address to a real memory address is depicted in figure 3.5-1. The algorithm to obtain the proper Page Descriptor in the system page table has been described in the previous subparagraphs.

The dotted lines indicate the variations in field lengths which are introduced by the variable page size.

3.6 ACCESS PROTECTION

The smallest unit of access protection which can be specified shall be a segment. Four mechanisms shall be provided to facilitate interprocess and intraprocess protections. (One mechanism for interprocess and three for intraprocess protection.) The interprocess protection shall be achieved by means of the process segment table which shall define the address space of a process. Segment Descriptor control fields shall be used to specify whether Read, Execute or Write access to a segment is permitted. The ring structure shall be used to organize the segments into a privileged hierarchy according to the ring number associated with each of the segments. The key/lock facility shall be used to partition the process access space into several subspaces with only restricted access from one to the other. When both ring and key/lock facilities are used, the process address space shall be organized with a vertical privileged hierarchy complemented by horizontal partitions.

3.6.1 Access Control Fields

The Execute, Read and Write access to each segment shall be controlled by the XP, RP and WP fields of each associated Segment Descriptor. The format and descriptions of the fields are specified in paragraph 3.3.1.1 of this specification.

3.6.2 Ring Hierarchy

The ability to grant access rights to a particular segment is not sufficient control, and that mechanism is augmented by a technique governing intraprocess control. This technique is an extension of the common two state (system state and user state) machines. The central processor operates in any of fifteen states (levels of privilege). These states are rings of protection. Ring one shall be the most privileged ring while ring 15 shall be the least privileged ring. Ring 0 (zero) has a special meaning as described in paragraph 2.8.1.13. In general, segments in the same ring have access to each other limited only by their prescribed access modes. However, communication between segments in different rings is carefully controlled. Passing control inwards (to a smaller ring number) is achieved by providing the callee with a gate through which the caller must pass. The most common example of this process occurs when a user calls on the operating system to perform a task. To ensure protection when returning from an outward call, both outward calls and inward returns shall result in interrupts and transfer of control to the operating system.

3.6.2.1 Execute Ring Bracket

It is frequently convenient to allow a segment to execute in several rings. This is accomplished by giving the segment an execute bracket. This bracket delimits the rings in which the segment may be executed, always provided that the segment has execute access granted by the XP field. The R1-R2 fields in the segment descriptor are used to denote the rings of which a segment may be a member.

Execute Access
R1 ≤ P.RN ≤ R2

If a process is executing in a ring contained in the execute bracket of a segment, and control is transferred to that segment, then the ring of execution is unchanged, (see the Branch instruction description in subparagraph 2.2.3.6 of this specification).

For the Call instruction, as described in subparagraph 2.6.1.2 of this specification, if the current ring of execution was greater than the ring bracket, it would be set equal to the greater ring number in the bracket, assuming the transfer of execution control is allowed.

3.6.2.2 Read and Write Limits

The concept of execute ring bracket is extended to read and write protection. A process must be executing within the read or write limit of a segment, and appropriate access must have been granted for their operations to be executed. The conditions for reading and writing a segment are given below.

Write Access	Read Access
PVA.RN ≤ R1	PVA.RN ≤ R2

Where the PVA.RN is the ring number contained in the A Register, the Keypoint Buffer pointer, the Trap pointer, the Debug List pointer, or the Top-of-Stack pointer with which the access is being made.

3.6.2.3 Call Ring Limit

When a procedure makes a call on another procedure executing in same or inner ring bracket, the right to make the call must first be validated, and the proper use of the gate must be checked. The authority to make the call has been given to the caller if it is entered via the proper entry points (gate), and if:

PVA.RN ≤ CB−R3 (Code Base Pointer − R3 field)

(See 2.5.5.1 for the Code Base Pointer format.)

To further assure that the call is not made to an outer ring bracket, a check on PVA.RN ≥ R1 is also made. The control of the call gate is implemented via the binding section which contains all the allowable entry points (code base pointers) to a procedure. The binding section is constructed by System Linker and is not modifiable by regular procedures. The format of code base pointer is described in section 2.5.5.

3.6.3 Key/Lock Facility

The Key/Lock is another protection facility that complements the ring hierarchy for controlling the intraprocess access. It can be used to partition procedures and/or data within the same ring bracket into zones with restricted access between each other. Functionally, the Key/Lock structure is an extension of conventional storage key structure. The unit of protection, however, has been changed from physical storage blocks into virtual segments.

The Key/Lock facility provides the following capabilities:

1. Restricted access to data to the procedure or procedures which own that data.
2. Write control within a ring bracket.
3. Write control of data in less privileged rings from more privileged rings.
4. Total isolation of data in less privileged rings from more privileged rings.

3.6.3.1 Format of Key/Lock Fields

The 8-bit field (bits 32-39) of the Segment Descriptor specifies the Lock for the associated segment. The format of this field is as follows:

32	33	34 39
0	1	LOCK

When the 6-bit Lock field (bits 34-39) is equal to zero, there is no Lock for the associated segment. When the Lock field is not equal to zero, it contains the 6-bit Lock for the associated field. Bits 32-33 affect only the Key/Lock compare and only on the initial 800 Series hardware implementations. When Key/Lock comparisons are not being made, the value of bits 32-33 have no effect on processor operation. When Key/Lock comparisons are to be made, the software must set bits 32-33 to 01. Failure to do so will cause initial 800 Series hardware to perform in a manner different from this Key/Lock description. Note that later hardware implementations may ignore bits 32-33.

There is a 6-bit Key associated with the P register as described in paragraph 2.1.1.1. When the 6-bit key field in the P register is equal to zero, the Key is considered to be a Master Key. When this field is not zero, it contains the 6-bit Key for the currently executing process.

3.6.3.2 Access Validations

The use of Keys and Locks are further controlled by the RP and WP fields in the Segment Descriptor (see paragraph 3.3.1.1).

Read/Write Access

For read or write accesses the Key comparison only occurs when Key/Lock control is specified for that type of access; i.e. RP=01 for read operations and WP=01 for write operations. The Key of the P register is compared with the Lock of the segment to be accessed. The operation proceeds only when:

- Key is equal to the Lock
- P register has a master key
- Lock is zero or no lock

An unsuccessful comparison (when the comparison is required via the RP or WP fields) causes detection of an access violation as specified in paragraph 2.8.1.7.

Call/Branch (2.6.1.2, 2.2.3.6)

For Call and Branch, the Key for the new P register is taken directly from the Lock field in the SDE associated with the new P address.

Return (2.6.1.4)

On a return operation, the Key for the new P register is obtained from the Stack Frame Save Area. The hardware shall compare this Key with the Lock in the SDE for the segment associated with the P register from the Stack Frame Save Area. If the two are not identical, an access violation shall be detected.

3.6.3.3 Software Conventions

It is expected that some software conventions will be followed for using the Key/Lock facility. The following are examples of such conventions.

- By using nonzero Locks, data can be restricted to be written or accessed by only "local" procedures. Normally, no procedure will have a Master Key.
- User and System procedures are normally assigned with a nonzero Key. All nonlocal data are not controlled.

4.0 CENTRAL MEMORY

4.1 GENERAL

Central memory shall provide main storage for all central processors in a system. It shall also provide the primary communication paths for these central processors. It shall also be accessible by IOU's and attached processors. These processors, IOUs and attached processors shall be able to access all central memory.

These accesses are illustrated in figure 4.1-1.

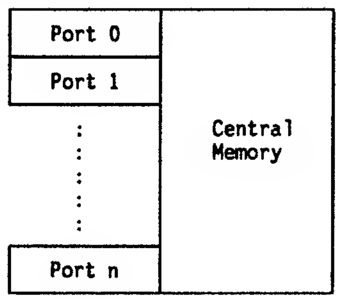


Figure 4.1-1. Memory System Elements

The ports support access for the central processors and/or for the IOUs and attached processors (see section 1.6). The ports supporting IOUs and attached processors shall conform to the Standard Memory Port definition in section 4.2. The ports supporting central processors may, but need not conform to the Standard Memory Port. The choice of port numbers is model dependent but numbers shall be recorded in section 1.6 to support software interfaces.

While the C180 architecture allows the selective disabling (flawing) of central memory pages, the page at RMA zero must be operable.

4.2 STANDARD MEMORY PORT INTERFACE

The standard memory ports shall be 64-bit ports and shall have a maximum transfer rate of 8 bytes per major clock cycle. These ports and all interfaces to these ports shall be synchronous with the local mainframe clock.

The standard ports on different systems shall be capable of accepting memory requests as described in section 1.6.

When the port is unable to accept additional requests, due to a memory bank busy or distributor busy, it shall send a PORTBUSY signal to the processor interface, thus stopping the flow of requests to the port. There shall be a sufficient buffering within the central memory port to allow for cable delay and processor recognition of the PORTBUSY signal. This delay shall not exceed 8 clock cycles; thus, up to 8 additional requests may have to be buffered within the port.

4.2.1 Interface Lines

Table 4.1-1 specifies the signals at a standard memory port.

Input into Standard Memory Port

Data In	64 lines + 8 lines (Parity)
Address*	28 lines + 4 lines (Parity)*
Mark Lines	8 lines + 1 line (Parity)
Tag In Lines	8 lines + 1 line (Parity)
Function Code	4 lines + 1 line (Parity)
Request	1 line

Output from Standard Memory Port

Data Out	64 lines + 8 lines (Parity)
Tag Out Lines	8 lines + 1 line (Parity)
Response Code	3 lines + 1 line (Parity)
Response	1 line
Port Busy	1 line
Interrupt	1 line

*See section 1.6 for a list of systems having less than 28 address bits.

Table 4.1-1. Standard Memory Port

a. Data In

The Data In lines shall contain the information which is to be stored into central memory during write operations. A parity bit shall accompany each byte of data.

The contents of the Data In lines on nonwrite operations shall be undefined but with correct parity.

b. Address

The address lines shall contain the word address that is to be accessed in memory. The contents of the address lines on Interrupt operations shall be undefined but with correct parity.

The RMA consists of RMA bits 33-60. The upper bits of this address (33-35) shall be supported by central memory as appropriate to the maximum central memory on the specific model. The address shall be accompanied by at least four parity bits.

The 32 bits in the Real Memory Address (RMA) are assigned as follows:

Bit	Description
32	This bit shall be ignored by the hardware and set to zero by the software.
33	Systems with less than or equal to 128 MByte maximum central memory Port Select per 2.10.1.1 Systems with greater than 128MB maximum central memory Select 1 of 2 GBytes
34	Select 512 of 1024 MBytes
35	Select 256 of 512 MBytes
36	Select 128 of 256 MBytes
37	Select 64 of 128 MBytes
38	Select 32 of 64 MBytes
↓	↓
61-63	Select 1 of 8 Bytes (not transmitted to the memory)

All models having either no C170 state or a central memory whose minimum degrade size is at least 16 MBytes may either ignore address bits above the installed central memory size or may report these as an error.

All models having C170 state and a minimum central memory including degrade which is less than 16 MBytes shall interpret addresses from the IOU or other external device as follows:

The address bits above those required for the maximum memory size on that model are ignored by the various memory models. Thus, the addresses are interpreted modulo "maximum memory."

References to addresses less than or equal to the maximum memory size but greater than the installed memory size, that is potential but nonexistent addresses, are interpreted as follows when the configuration switches are not active:

Read

When an attempt is made to read a memory cell which does not physically exist in real memory, memory will return a word consisting of all ones and will not give any error indication.

Write

When an attempt is made to write a memory cell which does not physically exist in real memory, the write completes as though the memory cell existed and no error indication is given.

c. Mark Lines

During partial write operations, these lines shall indicate which bytes are valid within the data in Word. One parity bit shall accompany the Mark lines.

The contents of the Mark lines on read type operations shall be undefined but with correct parity.

d. Tag In Lines

The Tag In lines shall contain requesting processor defined information during read or write operations. This tag information shall be returned unmodified to the requesting processor with the response from memory. If the requesting processor has more than one outstanding request, then it shall use this information for internal sequencing and routing of the response. Multiple requests to the same address shall be issued in the order they were received from the processor. A parity bit shall accompany the Tag In lines.

e. Function Code

The function lines shall contain the desired Function Code for a given memory request. Four lines shall specify up to sixteen functions. The function lines shall be accompanied by a parity bit. A detailed definition of the various functions is included in section 4.2.2 of this specification.

f. Request

This line shall be the strobe for all signals coming into the port.

g. Data Out

The Data Out lines shall contain the information being returned in response to a read operation. A parity bit shall accompany each byte of data.

On write type operations, the contents of the Data Out lines shall be undefined but with correct parity.

h. Tag Out Lines

The Tag Out lines shall contain a copy of the information placed on the Tag In lines during the read or write request. This information shall be used for sequencing as described in paragraph 4.2.1d. The tag + parity shall be returned to the processor exactly as it was received one major clock cycle prior to the corresponding data on the Data Out and Response Code lines.

i. Response

This signal shall provide the strobe for the Tag Out lines and shall occur one major clock cycle prior to the corresponding data on the Data Out and Response Code lines. Thus, for M2 or M3, this signal must be delayed for 56 or 64 ns respectively within the processor in order to subsequently serve as the strobe for the Data Out and Response Code lines from the memory ports.

j. Response Code

These lines shall specify the nature of the response being returned to the processor. These codes are described in detail in section 4.2.2.2. The Response Code shall be accompanied by a parity bit. Three lines specify up to eight response codes.

k. Interrupt

This line shall transmit an interrupt signal to the processor which is attached to the port. Section 4.2 describes how this signal is generated.

An interrupt from shared memory to the CMC of P3 (see figure 4.1-4) shall be transmitted to both port 0 and port 1 of the CMC.

l. Port Busy

This signal is described in paragraph 4.2.

4.2.2 Memory Functions, Responses, and Operations

4.2.2.1 Memory Functions

Memory shall perform the following operations as specified by the four bit code received on the function lines.

```
0000      READ
0001  *
0010      WRITE
0011  *

0100      READ AND SET LOCK
0101      READ AND CLEAR LOCK
0110      EXCHANGE
0111  *

1000  *
1001  *
1010      READ FREE RUNNING COUNTER
1011      REFRESH COUNTER RESYNC

1100      INTERRUPT
1101  *
1110  *
1111  *
```

* Function codes 1, 3, 7, 8, 9, D, E and F are ILLEGAL on a standard memory port and will result in a REJECT response. Function code A will also result in a REJECT response from the standard memory ports on M3 and THETA.

4.2.2.2 Memory Responses

The following response codes shall be sent to a processor in response to function codes.

```
000   WRITE RESPONSE
001   WRITE RESPONSE UNCORRECTABLE ERROR
010   WRITE RESPONSE CORRECTED ERROR
011   INTERRUPT RESPONSE

100   READ RESPONSE
101   READ RESPONSE UNCORRECTABLE ERROR
110   READ RESPONSE CORRECTED ERROR
111   REJECT
```

4.2.2.3 Memory Operations

4.2.2.3.1 Read

Initiation

- Function Code, Address and Tag are received during one clock period.

Response

- Response Code, Tag, and 64 bits of data as specified by the fullword address.

4.2.2.3.2 Write

This operation shall modify the bytes in the word specified by the word address and mark bits.

Initiation

- Function Code, Address, Tag, Mark Bits, and 64 bits of data are received during one clock period.

Response

- Response Code and Tag

If all Mark bits are set on a WRITE operation, the selected memory bank shall perform a Write Cycle. If any Mark Bits are cleared on a WRITE operation, the selected memory bank shall perform a read of the specified fullword, modify the bytes as specified by the mark bits and write the modified word into memory. No other accesses from any port shall be permitted to the central memory word from the beginning of the read to the end of the write.

4.2.2.3.3 Read and Set Lock; Read and Clear Lock; Exchange

These operations shall modify the bytes in the word specified by the word address and mark bits. Eight bytes of unmodified data shall be returned to the processor on these operations. No other accesses from any port shall be permitted to the central memory word from the beginning of the read to the end of the write.

The READ AND SET LOCK operation shall form a logical "OR" between the marked Data In bytes and the data read from memory, and shall rewrite the modified data into memory.

The READ AND CLEAR LOCK operation shall form a logical "AND" between the marked Data In bytes and the data read from memory, and shall rewrite the modified data into memory.

The EXCHANGE operation shall exchange the marked Data In bytes with the corresponding bytes in the word read from memory, and shall rewrite the modified data into memory.

Initiation

- Function Code, Address, Tag, Mark Bits, and data are received during one clock period.

Response

- Response Code, Tag, and 64 bits of data as specified by the fullword address.

4.2.2.3.4 Read Free Running Counter

Initiation

- Function Code, and Tags are received during one clock period.

Response

- Response Code, Tag, and 64 bits of data from the Free Running Counter. This shall consist of either:

  64 bits of counter, or

  48 bits of counter, right-justified, and zero-filled in the leftmost 16 bits.

On M3, this code is treated as an ILLEGAL function on ports 2 and 3 (ECS and IOU).

4.2.2.3.5 Refresh Counter Resync

This function is a hardware debug tool and, on memories having refresh counters, shall resynchronize the next request on this port with the memory refresh counter. (Refer to appropriate memory engineering specification.)

Initiation

- Function code and tag are received during one clock period. The address and data are ignored by the memory but must have correct parity.

Response

- Response code and tag. The response code returned shall be an Interrupt Response (011).

4.2.2.3.6 Interrupt

This function shall send an interrupt to the processor attached to the port specified by the contents of the data received on the Data In lines. Bits are assigned as follows:

Bit Number	60	61	62	63
Send Interrupt to Port No.	3	2	1	0

Bits 0-59 shall not be used to send interrupts. These bits shall be ignored by the memory but shall have correct parity.

Initiation

- Function and data are received during a single clock period.

Response

- A single Interrupt Response is returned.

One or more ports may receive an interrupt due to an interrupt operation. This includes sending an interrupt to one's self. The Interrupt signal is not required to accompany the Interrupt Response and may be significantly earlier. When interrupting one's self, the instruction which issued the interrupt must complete before the interrupt is taken.

4.2.2.4 Function and Response Code Interrelationships

Table 4.2-1 shows what types of conditions cause a particular response for a given function. Table 4.2-2 shows what hardware action is taken when a particular condition is sensed for a given function.

	Reject	Read Response Correctable Error	Read Response Uncorrectable Error	Read Response	Interrupt Response	Write Response Correctable Error	Write Response Uncorrectable Error	Write Response
Read	6	2	3,4	1	N/A	N/A	N/A	N/A
Write	6	N/A	N/A	N/A	N/A	2	3,4,5	1
Read and Set Lock	6	2	3,4,5	1	N/A	N/A	N/A	N/A
Read and Clear Lock	6	2	3,4,5	1	N/A	N/A	N/A	N/A
Exchange	6	2	3,4,5	1	N/A	N/A	N/A	N/A
Read Free Running Counter	6	N/A	4	1	N/A	N/A	N/A	N/A
Refresh Counter Resync	6	N/A	N/A	N/A	1	N/A	N/A	N/A
Interrupt	6,7	N/A	N/A	N/A	1	N/A	N/A	N/A

1) Normal transfer
2) Corrected error
3) Multiple bit error, OR
   Read parity error (applies when SECDED is disabled), OR
   Corrected error and Uncorrectable error on the same operation
4) Parity error (except Function code parity error or Tag-in parity error)
5) Bounds fault
6) Function code parity error OR Illegal function
7) Data-in parity error

Table 4.2-1. Function vs. Response Code for a Given Failure

Function	Illegal Function	Bounds Fault	Multiple Error & Read Parity Error*	Corrected Error	Function Parity Error	Tag Parity Error	Mark Parity Error	Address Parity Error	Data In Parity Error	Partial Write Path Parity Error
Read	2	N/A	3	3	2	3	3	3	3	N/A
Write	2	1	1	4	2	3	1	1	3	5
Read and Set Lock	2	1	1	4	2	3	1	1	1	5
Read and Clear Lock	2	1	1	4	2	3	1	1	1	5
Exchange	2	1	1	4	2	3	1	1	1	5
Read Free Running Counter	2	N/A	N/A	N/A	2	3	N/A	3	N/A	N/A
Refresh Counter Resync	2	N/A	N/A	N/A	2	N/A	N/A	N/A	N/A	N/A
Interrupt	2	N/A	N/A	N/A	2	3	N/A	N/A	2	N/A

* Applies when SECDED is disabled.

1) Inhibit Write
2) Force a single Reject response and inhibit Writes and Interrupts.
3) Operation proceeds normally with appropriate response.
4) The data read from memory shall be corrected, the marked bytes inserted, new SECDED code generated, the modified corrected data written back into memory.
5) Parity errors detected in the partial write paths will force a double error in the SECDED code rewritten into memory when the operation is a partial write operation.

Table 4.2-2. Hardware Action Taken for Function vs. Failure in Addition to the Responses Shown in Table 4.2-1

4.3 MEMORY PERFORMANCE REQUIREMENTS

4.3.1 Ports

The maximum transfer rate on the Memory port shall be eight bytes per major clock cycle.

4.3.2 Distribution

Distributor performance shall be model-dependent.

4.3.3 Access Time

Access time shall be model-dependent.

4.3.4 Bank Cycle Time

Bank Cycle Time shall be model-dependent.

4.4 RAM FEATURES (SEE ALSO SECTION 8)

4.4.1 Parity

All address, control, and data paths that constitute the port interfaces shall carry a parity bit for each eight bit byte. All major address, control, and data paths which are internal to the memory and do not carry SECDED code shall carry a parity bit for each eight bit byte.

4.4.2 Single Error Correction/Double Error Detection - SECDED

All data within the memory banks shall be protected with SECDED logic. It shall be possible to disable the SECDED logic by utilizing the Environment Control Register. If SECDED is disabled, byte parity shall be carried across from the ports and stored into the memory banks.

4.4.3 Noninterleaved Mode

Memory models may, but are not required to have the capability of operation in noninterleaved mode. Noninterleaved mode, when present, shall be controlled by the Environment Control Register. When in noninterleaved mode, sequential addresses shall reside in the same memory bank. This feature will allow software to degrade memory with a minimum impact on capacity.

4.4.4 Memory Configuration Switches

See section 1.6 for a list of systems containing this feature.

There shall be Memory Configuration switches within the memory to allow the logical reconfiguration of memory to remove failing memory portions from the address space. The available memory remaining after the reconfiguration shall be contiguous and start at address zero as seen from the processor.

In situations where a portion of the memory actually installed in a machine is 'wired down' (it is not addressable by the software), the configuration switches shall operate on the entire installed memory. In a 16MB system which is wired down to 8MB, for example, if the used 8MB fails it could be replaced by the formerly unused 8MB through use of the appropriate configuration switches.

There are two implementations of configuration switches. When bit 23 of the Memory OI register is clear, the description in 4.4.4.1 applies. When bit 23 is set, the description in 4.4.4.2 applies.

4.4.4.1 OI Bit 23 Clear (Three-position switches)

Switches SW1 through SW6 are three-position switches designated as follows:

	Switch Centered	Switch Up	Switch Down	OI Bit
SW1	No effect	RMA bit 38=1	RMA bit 38=0	17
SW2	No effect	RMA bit 39=1	RMA bit 39=0	18
SW3	No effect	RMA bit 40=1	RMA bit 40=0	19
SW4	No effect	RMA bit 41=1	RMA bit 41=0	20
SW5	No effect	RMA bit 42=1	RMA bit 42=0	21
SW6	No effect	RMA bit 43=1	RMA bit 43=0	22

Table 4.4-1. Memory Configuration Switches (three-position)

Bits 17-22 shall always be zero if the associated switch is not implemented on the specific memory model.

The following table lists the appropriate reconfiguration and the resulting values of bits 16-22 for specific failures for each memory increment available on M2/M3.

Installed Memory	Failing Portion of Memory RMA Bit						Memory Configuration Switches						Options Installed Req. Bits 16-22							Remaining Available Memory
	38	39	40	41	42	43	1	2	3	4	5	6	16	17	18	19	20	21	22	
1 MB																				No Degrade Available
2	x	x	x	x	x	0	-	-	-	-	-	U	1	0	0	0	0	0	1	1MB
	x	x	x	x	x	1	-	-	-	-	-	D	0	0	0	0	0	0	1	1
3	x	x	x	x	x	0	-	-	-	-	-	U	1	0	0	0	0	0	1	1
	x	x	x	x	0	1	-	-	-	-	-	0	0	0	0	0	0	0	1	1
	x	x	x	x	1	0	-	-	-	-	0	-	0	0	0	0	0	1	0	2
4	x	x	x	x	0	x	-	-	-	-	U	-	1	0	0	0	0	1	0	2
	x	x	x	x	1	x	-	-	-	-	0	-	0	0	0	0	0	1	0	2
5	x	x	x	0	0	x	-	-	-	-	U	-	1	0	0	0	0	1	0	2
	x	x	x	0	1	x	-	-	-	-	0	-	0	0	0	0	0	1	0	2
	x	x	x	1	0	1	-	-	-	0	-	-	0	0	0	0	1	0	0	4
6	x	x	x	0	0	x	-	-	-	-	U	-	1	0	0	0	0	1	0	2
	x	x	x	0	1	x	-	-	-	-	D	-	0	0	0	0	0	1	0	2
	x	x	x	1	0	x	-	-	-	0	-	-	0	0	0	0	1	0	0	4
7	x	x	x	0	x	x	-	-	-	U	-	-	1	0	0	0	1	0	0	3
	x	x	x	1	0	x	-	-	-	D	-	-	0	0	0	0	1	0	0	4
	x	x	x	1	1	0	-	-	-	0	-	-	0	0	0	0	1	0	0	4
8	x	x	x	0	x	x	-	-	-	U	-	-	1	0	0	0	1	0	0	4
	x	x	x	1	x	x	-	-	-	0	-	-	0	0	0	0	1	0	0	4
10	x	x	0	0	x	x	-	-	-	U	-	-	1	0	0	0	1	0	0	4
	x	x	0	1	x	x	-	-	-	0	-	-	0	0	0	0	1	0	0	4
	x	x	1	0	0	x	-	-	0	-	-	-	0	0	0	1	0	0	0	8
12	x	x	0	0	x	x	-	-	-	U	-	-	1	0	0	0	1	0	0	4
	x	x	0	1	x	x	-	-	-	0	-	-	0	0	0	0	1	0	0	4
	x	x	1	0	x	x	-	-	0	-	-	-	0	0	0	1	0	0	0	8
14	x	x	0	x	x	x	-	-	U	-	-	-	1	0	0	1	0	0	0	6
	x	x	1	0	x	x	-	-	0	-	-	-	0	0	0	1	0	0	0	8
	x	x	1	1	0	x	-	-	0	-	-	-	0	0	0	1	0	0	0	8
16	x	x	0	x	x	x	-	-	U	-	-	-	1	0	0	1	0	0	0	8
	x	x	1	x	x	x	-	-	0	-	-	-	0	0	0	1	0	0	0	8

- = Switch in Center
U = Switch Up
0 = Switch Down

Table 4.4-2. Memory Reconfigurations

Further reconfiguration may be performed by setting (or clearing) additional switches. For example, a 14 MB memory reconfigured to 6 MB as indicated in the table may be further reconfigured to a 2 or 4 MB memory by using the additional switch indicated under the 6 MB entry.

The amount of available memory after reconfiguration is the amount associated with the rightmost one bit of bits 17-22 as follows (with the exception noted below):

Bit 17	*32 MB*
18	*16 MB*
19	*8 MB*
20	*4 MB*
21	*2 MB*
22	*1 MB*

There are two exceptions which require examination of bit 16; 14 MB degraded to 6 MB and 7 MB degraded to 3 MB.

After reconfiguration, addresses greater than the remaining available memory will either "wrap-around" into available memory or reference nonexistent memory (see 4.1.7) as specified by the configuration switches. For example, consider a 5 MB memory reconfigured as shown below:

	Physical Memory Bank Referenced			
Processor Address (RMA Bits 41-43	With No Reconfiguration	Failure in 000 or 001 SW5 Up	Failure in 010 or 011 SW5 Down	Failure in 100 SW4 Down
000	000	010	000	000
001	001	011	001	001
010	010	010	000	010
011	011	011	001	011
100	100	Nonexistent	100	000
101	Nonexistent	Nonexistent	Nonexistent	001
110	Nonexistent	Nonexistent	100	010
111	Nonexistent	Nonexistent	Nonexistent	011

4.4.4.2 OI Bit 23 Set (Two-position switches)

Switches SW0 through SW6 are two-position switches designated as follows:

	Switch Down	Switch Up	OI Bit
SW0	No effect	Invert RMA bit 36	16
SW1	No effect	Invert RMA bit 37	17
SW2	No effect	Invert RMA bit 38	18
SW3	No effect	Invert RMA bit 39	19
SW4	No effect	Invert RMA bit 40	20
SW5	No effect	Invert RMA bit 41	21
SW6	No effect	Invert RMA bit 42	22

Table 4.4-3. Memory Configuration Switches (two-position)

Bits 16-22 will always be zero if the associated switch is not implemented on the specific memory model.

4.5 MAINTENANCE REGISTERS

Table 4.5-1 lists memory maintenance registers and their access privilege. Maintenance registers fall into two classifications, those accessible via the Maintenance Access, and those accessible via memory ports.

Register Number	Mem. Port Access Privilege		Maintenance Channel Access	Register Name
	Read	Write		
00	No Access	No Access	Read	Status Summary
10	No Access	No Access	Read	Element ID
12	No Access	No Access	Read	Options Installed
20	No Access	No Access	Read/Write	Environment Control
21	No Access	No Access	Read/Write	Bounds Register
A0	No Access	No Access	Read/Write	Corrected Error Log Dis 0
A1 *	No Access	No Access	Read/Write	Corrected Error Log Dis 1
A2 *	No Access	No Access	Read/Write	Corrected Error Log Dis 2
A3 *	No Access	No Access	Read/Write	Corrected Error Log Dis 3
A4	No Access	No Access	Read/Write	Uncorrectable Error Log 1 Dis 0
A5 *	No Access	No Access	Read/Write	Uncorrectable Error Log 1 Dis 1
A6 *	No Access	No Access	Read/Write	Uncorrectable Error Log 1 Dis 2
A7 *	No Access	No Access	Read/Write	Uncorrectable Error Log 1 Dis 3
A8	No Access	No Access	Read/Write	Uncorrectable Error Log 2 Dis 0
A9 *	No Access	No Access	Read/Write	Uncorrectable Error Log 2 Dis 1
AA *	No Access	No Access	Read/Write	Uncorrectable Error Log 2 Dis 2
AB *	No Access	No Access	Read/Write	Uncorrectable Error Log 2 Dis 3
B0 †	Unprivileged	No Access	Write	Free Running Counter

* Since M2 and M3 have only one distributor, these registers do not exist in either of these memories.

† No address is necessary for the register accessed via the memory ports; this register is read unconditionally with a Read Free Running Counter function code.

Table 4.5-1. Memory Register Access Privileges

4.5.1 Maintenance Registers Accessible by the Maintenance Access

Maintenance registers shall provide the error and environment status information required in section 9.0. Write and read maintenance access is model dependent. Although not required, usage of the same register names and numbers is preferred to maintain commonality. Read and write maintenance access shall provide the following capability:

Read

The memory maintenance registers (table 4.5-1) may be read at any time.

Write

The memory maintenance registers in table 4.5-1 that have write access may be written at any time with the following exceptions:

- Environment Control Register (20)

  The following two bits may be switched at any time by performing a single write operation. Any one or both of these two bits may be altered:

  Disable Corrected Error Log
  Disable Corrected Error Response (Proposed)

  No other bits may be switched without halting memory activity except as specifically permitted in the model-dependent engineering specification.

- Bounds Register (21)

  Writing the bounds register without first halting all write activity on the port(s) either currently selected or to be selected by the write into the bounds registers shall cause undefined results with respect to the bounds checking.

- Uncorrectable Error Log (A4, A8)

  The error logs may be written at any time. The clear of an error log following the logging of an error is intended to be accomplished by a write operation. Writes into error logs which clear the valid bit and require further write operations to complete the process shall cause the contents of the error log to be undefined unless all memory references are halted for the duration of the write.

4.5.2 Maintenance Registers Accessible by Memory Ports

4.5.2.1 Free Running Counter

The Free Running Counter shall be either a 48-bit or a 64-bit incrementor. Bit 63 shall increment at a one microsecond rate. Successive reads of the Free Running Counter shall be guaranteed different values.

The Free Running Counter is read accessible via the memory port when a processor executes a Copy Free Running Counter to Xk instruction (Op. 08). No register address is necessary for access of this register. It is unconditionally read with a Read Free Running Counter function code.

The Maintenance Processor shall be able to write this counter at any time. Maintenance method shall be model dependent.

4.6 BOUNDS REGISTER

A Bounds Register defines a memory protection range either from the bottom of real memory to the bounds address or from the bounds address to the top of real memory.

A uniform model-independent register format is defined by the following guidelines and requirements. The register content is model-dependent based on the product supported memory size. All new design systems must satisfy these stated requirements. See section 1.6 for model-dependent differences on older systems.

1) A single bounds address with an Upper Memory Select bit is defined by the Bounds Register. The Upper Memory Select bit affects the corresponding port in the following way:

- If the Upper Memory Select bit is set, writes shall be inhibited for all addresses less than or equal to the bounds address.
- If the Upper Memory Select bit is clear, writes shall be inhibited for all addresses greater than or equal to the bounds address.

2) The bounds address must be on page boundaries.

3) The maximum size real memory provided by this product shall be covered by the Bounds Register address field.

4) When a processor utilizes different memories (external shared memories, or one or more central memories), a separate Bounds Register must exist for each memory.

5) When a write is inhibited, an Uncorrectable Error Response shall be returned and an entry shall be placed in the Uncorrectable Error Log.

6) Read operations are not tested for bounds.

7) Systems supporting concurrent maintenance must have a separate Bounds Register per element.

8) If a common Bounds Register is implemented, a Bit Vector is required that specifies those ports for which bounds checking is to be performed.

9) On-line access to modify either the Upper Memory Select bit or the Port Bounds Bit Vector must be provided. These control functions may be integrated into the Bounds Register.

5.0 INPUT/OUTPUT UNIT

5.1 GENERAL

The Input/Output Unit (IOU) provides communication capability between the CYBER 180 mainframe system and external devices. This communication is composed of data transfers between the external devices and areas within central memory.

The peripheral processors (PP) communicate with the central processor via central memory data structures and a processor interrupt mechanism. They communicate with external devices and each other over the I/O channels. The actual concurrent throughput of these channels is dependent on the available bandwidth of the attached memory. Frequently, systems are configured with redundant channel connections for fault tolerant requirements, and these are not used concurrently.

An option on some systems is the attachment of a second IOU to the second standard memory port in place of a CYBERPLUS or MAP V. See Table 1.5-2 for a list of systems supporting the optional second IOU.

The primary IOU radial interface maintenance channels will connect to all the system elements except to the secondary IOU. The secondary IOU radial interface ports are not used.

A CC598B console (IBM PC compatible computer) is required to support initialization, operating system displays, and maintenance of dual I4 systems. This replaces the CC634B console.

On the I4C, a dedicated load device load device CC598A console (IBM PC compatible computer) is required. This console is an extension of the CC598B console with the added functionality of acting as a file server to the IOU for common disk area data. On the I4C, all initialization and off-line maintenance software resides on the console disk.

A CC545 Display Console is required for NOS/BE operation, and is optionally supported for NOS operation.

A PP executes programs alone or in conjunction with other PPs to effect an orderly transfer of data between external devices and central memory. These programs are referred to as I/O drivers. The I/O drivers interact with requests placed in central memory by the operating system.

The I/O drivers use PP memory as a buffer for the data transfer between the external devices and central memory. This buffering ensures that the data transfer is isolated from variations in the central memory transfer rate. In addition, the I0 PPs support a direct memory access (DMA) instruction which transfers data directly between central memory and an I/O channel.

5.2 PERIPHERAL PROCESSOR

5.2.1 Organization

The PP has four main, central components - memory, arithmetic register, arithmetic unit, address and relocation registers.

5.2.1.1 Memory

Units of information in memory are referred to as words (16 bits). Words are numbered consecutively from zero, and bits within a word are numbered left to right starting at bit 48 as indicated below:

```
4    5              6
8    2              3
+----+---------------+
|    |               |
+----+---------------+
```

On I1, I1CR, and I2, the random access memory contains 4096 words; this is true for both *C170 state* and C180 state operations. All models of the I4 contain 8192 words; *for C170 state operations, however, only 4096 words are normally accessible on the I4.* The I0 contains 16384 words. The CYBER 170 12-bit words are contained in bits 52-63 of the memory word, with bits 48-51 cleared.

5.2.1.2 Arithmetic Register

The 18-bit arithmetic register (A-register) is used as one of the instruction operands and for the result of arithmetic and logical instructions. It also provides the least significant 18 bits of the central memory address for the cent instructions and the word count for I/O instructions. Bits within this register are numbered left to right, beginning with bit position 46 as follows:

```
4                    6
6                    3
+--------------------+
|      18 bits       |
+--------------------+
```

5.2.1.3 Arithmetic Logic Unit

The arithmetic logic unit (ALU) performs 18-bit ones complement arithmetic and 18-bit logical and shift operations. Any overflow of 18 bits is ignored.

5.2.1.4 Address registers

On I1 and I2, the *12-bit* program address register (P-register) provides the address the instruction being executed. This register can increment throughout all possible PP memory locations (0-7777, binary). Bits within this register are numbered left to right, beginning with bit position 52 as follows:

```
5              6
2              3
+---------------+
|    12 bits    |
+---------------+
```

On the I0 and I4, the program address register is a 16-bit register that can address PP locations 0-17777 binary (I4), and 0-37777 binary (I0).

```
4                    6
8                    3
+---------------------+
|       16 bits       |
+---------------------+
```

The relocation register (R-register) is used in conjunction with the A-register to form an absolute central memory address used by the central memory read/write instructions. The R-register is cleared only by a power-on master clear and by issuance of an IOU Maintenance Access Control Function #6 (Master Clear ADU), (see 5.6.1). A PP deadstart does not alter the contents of the R-register. The length of the R-register is model dependent. See section 5.2.2.10 for a description of usage of this register.

The I4 NIO PPs support a 4KW mode in which the PPs reference memory in a compatible manner to the I2. This mode (4KW) is the default mode and is deselected by setting a bit in an IOU maintenance register.

5.2.2 Instruction Set

The CYBER 180 PP instruction set is based on the CYBER 170 PP instruction set. The 7-bit operation code includes the 6-bit operation code of the CYBER 170 PP. Extensions to the instruction set allow programs to manipulate 16-bit PP words, 64-bit central memory words as both 12-bit and 16-bit bytes, and to reference 28-bit central memory addresses. Appendix E contains a list of all PP instructions.

5.2.2.1 Instruction Formats

All PP instructions are represented in one of four formats. Two of these use a single 16-bit word; the other two use two consecutive 16-bit words. These formats are represented below.

16-bit formats

48		52	58	63
g	0 0 0	f	d	

g	0 0 0	f	s	c

32-bit formats

48		52	58	63
g	0 0 0	f	d	
0 0 0 0		m*		

g	0 0 0	f	s	c
0 0 0 0		m*		

* Note that m is expanded to 16 bits (48-63) for I/O function codes on CYBER 16-bit channels and for address specification on I0 and I4 (8KW mode) PPs.

The following field descriptions apply to both instruction formats.

- **f** 6 bits; the least significant 6 bits of the 7-bit operation code.
- **d** 6 bits ; an operand, part of an operand, or an address specification depending upon the instruction.
- **c** 5 bits; a channel number.
- **m** 12 bits; part of an operand, an address specification or an I/O function code depending upon the instruction.

  Note that m is expanded to 16 bits (48-63) for I/O function codes on CYBER 16-bit channels and for address specification on I0 and I4 (8KW mode) PPs.
- **g** 1 bit; the most significant bit of the 7-bit operation code. In many instructions, g acts to control the width of the operand read from PP memory. If g is 0, the operand is 12 bits; if g is 1, the operand is 16 bits.
- **s** 1 bit; a sub-operation code used with certain I/O instructions.
- **0** unused bits reserved for future use which should be set to zero.

The CYBER 180 PP instruction set in these formats includes the CYBER 170 PP instructions as a subset, and therefore allows the execution of CYBER 170 PP programs without change in binary format. Exceptions are noted below:

1) The CYBER 170 PP instruction 27 x (read program address) operates as a keypoint instruction on the CYBER 180 IOU.
2) The CYBER 170 24 d and 25 d Pass instructions, only execute as passes on the CYBER 180 IOU providing the d-field is zero. Otherwise they are used to load and store the R-Register.
3) The CYBER 170 64 s c m, 65 s c m, 66 s c m, and 67 s c m instructions execute in the same manner on the CYBER 180 IOU when the s-field is clear. When s is set, the 64 and 65 instructions are used on the CYBER 180 IOU to test and set the channel flag bit while the 65 and 66 instructions are used to test the channel error flag bit.

5.2.2.2 Address Modes

Instruction operands are determined by the address mode of the instruction. Operands are available either from the instruction or from memory locations specified by the instruction. The operands may be 5, 6, 8, 12, 16 or 18 bits in length. Appendix F contains a list of PP instruction address modes.

5.2.2.2.1 No-Address Mode

The no-address mode uses the d-field directly as a 6-bit operand.

5.2.2.2.2 Constant Mode

The constant mode uses the d-field and the m-field directly as a 18-bit operand. The d-field contains the most significant 6 bits of the operand; the 12 bits of the m-field contain the least significant 12 bits of the operand.

5.2.2.2.3 Direct Mode

The direct mode uses the d-field as the 6-bit address of a 12-bit or 16-bit operand in memory. (See 5.2.2.1.)

5.2.2.2.4 Indirect Mode

The indirect mode uses the d-field as the 6-bit address of a word in memory that is used as the address of a 12-bit or 16-bit operand in memory. (See 5.2.2.1.) Thus, d specifies the operand indirectly.

5.2.2.2.5 Memory Mode

The memory mode uses the d-field and m-field to specify the address of a 12-bit or 16-bit operand in memory.

NOTE

Throughout section 5.2.2.2.5, replace the number 7777 with 17777 for I4 PPs in 8KW mode, and replace the number 7777 with 37777 for all I0 PPs.

If the d-field is 0, bits 52-63 of the m-field are used as the address.

If the d-field is not 0, the d-field is the 6-bit address of a 12-bit index. This index is added to bits 52-63 of the m-field to generate the 12-bit address of all possible PP memory locations (0 to 7777, binary).

The 12-bit address is specified by d and m (expressed in octal) as follows:

	m=0	m=7777	0<m<7777
d=0	0	0*	m
d≠0, (d)=0	0	0*	m
d≠0, (d)=7777	0	7777	m
d≠0, 0<(d)<7777	(d)	(d)	m+(d)

* Replace with 17777 for I4 PPs in 8KW mode and 37777 for all I0 PPs.

In the block I/O and central memory access instructions, d has an alternate meaning and is not used in address computation. The first word address for these instructions is formed directly from m and can reference location 7777. The order of reference is 7777-0000-0001-0002.

5.2.2.3 Nomenclature

Nomenclature used in the following instruction description is defined below:

A	Refers to the A-register (arithmetic register)
(A)	Refers to the contents of the A-register
X	Refers to a one bit field which can be either 0 or 1.
c	Refers to a 5-bit channel number
d	Refers to the value of the 6-bit d-field (no-address mode)
dm	Refers to the 18-bit value obtained from the d-field and the m-field (constant mode)
(d)	Refers to the contents of the location specified by the d-field (direct mode)
((d))	Refers to the contents of the location specified by the contents of the location specified by the d-field (indirect mode)
m+(d)	Refers to the address specified by the m-field indexed by the contents of the location specified by the d-field
(m+(d))	Refers to the contents of the location specified by the m-field indexed by the contents of the location specified by the d-field (memory mode)
P	Refers to the P-register(program address register)
R	Refers to the R-register(relocation register)
(R)	Refers to the contents of the R-register
(R)+(A)	Refers to the central memory address formed from the contents of the R- and A-registers as described in paragraph 5.2.2.10 of this specification.

5.2.2.4 General Instruction Notes

5.2.2.4.1 Short Word (12-Bit) Stores

The instructions which store the least significant 12 bits of a PP memory word (0002, 0034-0037, 0044-0047, 0054-0057) and the central reads (0060, 0061) all clear the most significant 4 bits of the PP memory word.

5.2.2.4.2 Usage of PP Location 0 During Instruction Execution

The four central memory block transfer instructions (0061, 0063, 1061, 1063) and the four block input/output instructions (0071, 0073, 1071, 1073) all make use of memory location 0 to save the value of the P-register during instruction execution. This allows the P-register and associated incrementing logic to function as the PP memory address for the block transfers. The actual value stored in location 0 is the address of the m-field of the instruction, i.e. P+1. When the instruction exits, the contents of location 0 is incremented by 1 and placed in the P-register before the next instruction is executed, allowing normal instruction sequencing to resume.

If the contents of location 0 are altered during the instruction execution, the next instruction will be executed out of the initial sequence at (0)+1. This action could result from the block transfer of data from central memory or a channel into location 0. As such, this action may or may not be intended.

5.2.2.4.3 Unused Bits

When one or more bits from an instruction are unused, i.e., their value(s) and associated function(s) are not specified within the instruction description, the execution of these instructions shall not be affected by the values of these bits. However, it is recommended that such bits be set equal to zero.

5.2.2.4.4 Compass Mnemonic

The bracketed expression following such instruction code is the compass assembler mnemonic operation code.

5.2.2.4.5 I0 and I4 PPs

The following instruction definitions describe a 4K word PP IOU implementation. On the I4 all memory references refer to a 8KW PP memory (except for the I4 NIO PPs when in 4KW mode). On the I0 PPs all memory references refer to a 16KW PP memory.

5.2.2.4.6 I0 Instruction Usage Restriction

The I0 PP has a pipeline architecture that is one instruction deep. As one instruction is executed, the next memory location is read to obtain the next instruction. In the I0, therefore, it is impossible to use a PP instruction to modify the memory location containing the next instruction to be executed.

5.2.2.4.7 I0 Interrupt Processor Instruction

The 930/932 requires that bank 0 is accessed during the Interrupt Processor instruction. This places the requirement on PP programs to clear the five least significant bits of the A register prior to an Interrupt Processor instruction. (See section 5.2.2.12.)

5.2.2.5 Load and Store

The load and store instructions provide the means to transfer 6-bit, 12-bit, 16-bit and 18-bit quantities between A and memory.

Load d 0014 d [LDN d]

This instruction enters a copy of the 6-bit d-field into the A-register; d is considered to be a positive integer which is loaded into bits 58-63 of the A-register, with bits 46-57 of A cleared.

Load complement d 0015 d [LCN d]

This instruction enters a complemented copy of the 6-bit d-field into the A-register; bits 46-57 of A are set to one, and bits 58-63 are bit-by-bit complements of the corresponding bits in the d-field.

Load dm 0020 d m [LDC dm]

This instruction clears the A-register and enters an 18-bit operand consisting of the 6-bit d-field and the 12-bit m-field into bits 46-51 and bits 52-63, respectively, of the A-register.

Load (d) 0030 d [LDD d]

This instruction clears the A-register and enters a 12-bit quantity from bits 52-63 of location d, treated as a 12-bit positive integer, into bits 52-63 of A. Bits 46-51 of A are cleared.

Load (d) long 1030 d [LDDL d]

This instruction clears the A-register and enters a 16-bit quantity from location d, treated as a 16-bit positive integer, into bits 48-63 of A. Bits 46-47 of A are cleared.

Store (d) 0034 d [STD d]

This instruction stores the 12-bit quantity in bits 52-63 of the A-register into location d. Bits 48-51 of location d are cleared. The content of A is not altered.

Store (d) long 1034 d [STDL d]

This instruction stores the 16-bit quantity in bits 48-63 of the A-register into location d. The content of A is not altered.

Load ((d)) 0040 d [LDI d]

This instruction clears A and loads bits 52-63 of ((d)) into bits 52-63 of A. Bits 46-51 of A are cleared.

Load ((d)) long 1040 d [LDIL d]

This instruction clears A and loads ((d)) into bits 48-63 of A. Bits 46-47 of A are cleared.

Store ((d)) 0044 d [STI d]

This instruction stores bits 52-63 of the A-register into bits 52-63 of the storage location specified by the content of location d. Bits 48-51 of ((d)) are cleared. The content of A is not altered.

Store ((d)) long 1044 d [STIL d]

This instruction stores bits 48-63 of the A-register into the storage location specified by the content of location d. The content of A is not altered.

Load (m+(d)) 0050 d m [LDM m,d]

This instruction clears the A-register and enters bits 52-63 of a 12-bit operand from storage into bits 52-63 of A. The address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the operand address is given by m. The operand enters A as a 12-bit positive integer and bits 46-51 of A are cleared. (See section 5.2.2.2.5 for a description of address formation.)

Load (m+(d)) long 1050 d m [LDML m,d]

This instruction clears the A-register and enters a 16-bit operand from storage into bits 48-63 of A. The address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the operand address is given by m. The operand enters A as a 16-bit positive integer and bits 46-47 of A are cleared. (See section 5.2.2.2.5 for a description of address formation.)

Store (m+(d)) 0054 d m [STM m,d]

This instruction stores bits 52-63 of the A-register into bits 52-63 of storage. Bits 48-51 of (m+(d)) are cleared. The storage address is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. The content of A is not altered. (See section 5.2.2.2.5 for a description of address formation.)

Store (m+(d)) long 1054 d m [STML m,d]

This instruction stores bits 48-63 of the A-register. The storage address is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. The content of A is not altered. (See section 5.2.2.2.5 for a description of address formation.)

5.2.2.6 Arithmetic

The arithmetic instructions perform integer arithmetic with the contents of A as one operand and the other operand as specified by the instruction. The result replaces the original contents of A. The operands are considered as ones complement integers and the arithmetic is performed in ones complement mode.

Add d 0016 d [ADN d]

This instruction adds the d-field, considered as a 6-bit positive quantity, to the current content of the A-register. (See 5.2.2.1) The result remains in A. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the d-field by filling the remaining higher-order bits with zeros.

Subtract d 0017 d [SBN d]

This instruction subtracts the d-field, considered as a 6-bit positive quantity, from the current content of the A-register. (See 5.2.2.1) The result remains in A. An 18-bit operand forms from the d-field by inserting bit-by-bit complements of the corresponding d-field bits in the lower order bit positions, and by filling the remaining higher order bits with ones. This 18-bit operand adds to the original content of A in an 18-bit ones complement mode.

Add dm 0021 d m [ADC dm]

This instruction adds an 18-bit operand consisting of the d- and m-fields to the current content of the A-register. The result remains in A. The addition is in an 18-bit ones complement mode. The d-field forms the six highest order bits, and the m-field completes the twelve lowest-order bits.

Add (d) 0031 d [ADD d]

This instruction adds bits 52-63 of location d, considered as a 12-bit positive integer, to the current content of the A-register. The result remains in A. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from bits 52-63 of location d by adding six higher-order zero bits.

Add (d) long 1031 d [ADDL d]

This instruction adds the content of location d, considered as a 16-bit positive integer, to the current content of a A-register. The result remains in A. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from location d by adding two higher-order zero bits.

Subtract (d) 0032 d [SBD d]

This instruction subtracts bits 52-63 of location d, considered as a 12-bit positive integer from the current quantity in the A-register. The result remains in A. The operation adds the complement of location d to the content of A in an 18-bit ones complement mode. An 18-bit operand forms from location d. This operand consists of six one bits in the highest-order bit positions, and twelve lowest-order bits which are bit-by-bit complements of the corresponding bits in location d.

Subtract (d) long 1032 d [SBDL d]

This instruction subtracts the content of location d, considered as a 16-bit positive integer from the current quantity in the A-register. The result remains in A. The operation adds the complement of location d to the content of A in an 18-bit ones complement mode. An 18-bit operand forms from location d. This operand consists of two one bits in the highest-order bit positions, and sixteen lowest-order bits which are bit-by-bit complements of the corresponding bits in location d.

Add ((d)) 0041 d [ADI d]

This instruction adds a 12-bit operand considered as a 12-bit positive integer, from bits 52-63 of storage to the current quantity in the A-register. The result remains in A. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the 12-bit operand by adding six higher-order zero bits. The address for the operand is in location d.

Add ((d)) long 1041 d [ADIL d]

This instruction adds a 16-bit operand, considered as a 16-bit positive integer to the current quantity in the A-register. The result remains in A. The addition is in an 18-bit ones complement mode. An 18-bit operand forms the 16-bit operand by adding two higher order zero bits. The address for the operand is in location d.

Subtract ((d)) 0042 d [SBI d]

This instruction reads an operand from bits 52-63 of storage and subtracts it from the current contents of the A-register. The result remains in A. The address for the operand is in location d. The operation performs by adding the complement of the operand to the content of A in an 18-bit ones complement mode. An 18-bit operand for the addition forms from the 12-bit storage operand by forcing each of the highest-order six bits to a one value. The lowest-order 12 bits are the bit-by-bit complement of the storage operand values.

Subtract ((d)) long 1042 d [SBIL d]

This instruction reads an operand from storage and subtracts it from the current content of the A-register. The result remains in A. The address for the operand is in location d. The operation performs by adding the complement of the operand to the content of A in an 18-bit ones complement mode. An 18-bit operand for the addition forms from the 16-bit storage operand by forcing each of the highest-order two bits to a one value. The lowest-order 16 bits are the bit-by-bit complement of the storage operand values.

Add (m+(d)) 0051 d m [ADM m,d]

This instruction reads an operand from bits 52-63 of storage and adds it to the current content of the A-register. The result remains in A. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the 12-bit storage operand by adding six highest-order zero bits. The storage address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Add (m+(d)) long 1051 d m [ADML m,d]

This instruction reads an operand from storage and adds it to the current content of the A-register. The result remains in A. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the 16-bit storage operand by adding two highest-order zero bits. The storage address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Subtract (m+(d)) 0052 d m [SBM m,d]

This instruction reads an operand from bits 52-63 of storage and subtracts it from the current content of the A-register. The result remains in A. The operation performs by adding the complement of the storage operand to the content of A in an 18-bit ones complement mode. An 18-bit operand forms from the 12-bit storage operand by forcing each of the highest-order six bits to a one value. The lowest-order 12 bits are the bit-by-bit complement of the storage operand values. The storage address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Subtract (m+(d)) long 1052 d m [SBML m,d]

This instruction reads an operand from storage and subtracts it from the current content of the A-register. The result remains in A. The operation performs by adding the complement of the storage operand to the content of A in an 18-bit ones complement mode. An 18-bit operand forms from the 16-bit storage operand by forcing each of the highest-order two bits to a one value. The lowest-order 16 bits are the bit-by-bit complement of the storge operand values. The storage address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

5.2.2.7 Logical

The logical instructions perform operations with the contents of A as one operand and the other operand as specified by the instruction. The result replaces the original contents of A.

Shift d 0010 d [SHN d]

This instruction shifts the content of the A-register either to the right open-ended or to the left circularly as specified by the d-field. The d-field is treated as a 6-bit ones complement number. If the most significant bit in the d-field is zero, then the content of A shifts circularly to the left by the number of bit positions indicated in the value of the d-field. If the most significant bit in the d-field is one, the content of A shifts open-ended to the right by the complement of the value of the d-field.

In a left circular shift, the content of A shifts one bit position at a time. In each shift, the least significant bit position in the register fills by the bit previously held in the most significant bit position. Bits are not lost in this process but are repositioned toward the left. A d-field of zero causes no shift. A d-field in the range of 18 to 31 (decimal) causes a shift completely around the register, resulting in a shift of d−18 (decimal).

In a right open-ended shift, the content of A shifts one bit position at a time toward the less significant bit positions in the register. The most significant bit position in A fills with a zero value as each shift occurs. The least significant bit in A discards as each shift occurs. A maximum of 31 (decimal) shift counts may be used. For all shift counts larger than 17 (decimal), the final A-register value is 000000 (octal). A d-field of 77 (octal) causes no shift to take place.

Shift (d) 1010 d [SHDL d] **(IO only)**

This instruction shifts the contents of the A-register either to the right open-ended or to the left circularly as specified by the contents of bits 58-63 of location d. This field is treated as a 6-bit ones complement shift field number. If the shift field most significant bit is zero, then the content of A shifts circularly to the left by the number of bit positions indicated in the value of the shift field. If the most significant bit in the 6-bit shift field is one, the content of A shifts open-ended to the right by the complement of the value of the shift field.

In a left circular shift, the least significant bit position in the register (bit 2^0) fills by the bit previously held in the most significant bit position (bit 2^{17}). Bits are not lost in this process but are repositioned toward the left. A shift field of zero causes no shift. A shift field in the range of 18 to 31 (decimal) causes a shift completely around the register, resulting in a shift of d−18 (decimal).

In a right open-ended shift, the most significant bit position in A (bit 2^{17}) fills with a zero value as each shift occurs. The least significant bit in A (bit 2^0) discards as each shift occurs. A maximum of 31 (decimal) shift counts may occur. For all shift counts larger than 17 (decimal), the final A-register value is 000000 (octal). A shift field of 77 (octal) causes no shift to take place.

Logical difference d 0011 d [LMN d]

The logical difference instruction performs the logical difference function of d, considered as a 6-bit positive integer, and the corresponding bits of the A-register. Bits 58-63 of A are affected and bits 46-57 of A are not altered.

The logical difference function forms the bit-by-bit logical operation on two operands according to the following example:

operand 1	0011
operand 2	0101
result	0110

Logical product d 0012 d [LPN d]

The logical product instruction performs the logical product function of d, considered as a 6-bit positive integer, and the corresponding bits of the A-register. Bits 58-63 of A are affected and bits 46-57 of A are not altered.

The logical product function forms the bit-by-bit logical operation on two operands according to the following example:

operand 1	0011
operand 2	0101
result	0001

Selective clear d 0013 d [SCN d]

This instruction clears any bit in the A-register where the corresponding bit in the 6-bit d-field is a one. Bits 58-63 of A are affected and bits 46-57 of A are not altered.

Logical product dm 0022 d m [LPC dm]

This instruction forms the logical product of the content of the A-register and an 18-bit operand consisting of the d- and m-fields. The result remains in A. The d-field forms the highest-order six bits, and the m-field completes the lowest order 12 bits of the 18-bit operand. The logical product forms as described in the truth tables given for the 0012 instruction.

Logical product (d) long 1022 d [LPDL d]

This instruction forms, in the A-register, the logical product of the contents of location d, considered as a 16-bit quantity and the original contents of A. Bits 46-47 of A are cleared by this operation. The logical product forms as described in the truth tables given for the 0012 instruction.

Logical product ((d)) long 1023 d [LPIL d]

This instruction forms the logical product of a 16-bit operand read from storage and the original contents of the A-register. Bits 46-47 of A are cleared. The storage address for the operand is in location d.

Logical product (m+(d)) long 1024 d m [LPMLm,d]

This instruction forms the logical product of a 16-bit operand read from storage and the original contents of the A-register. Bits 46-47 of A are cleared. The address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero then the address of the operand is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Logical difference dm 0023 d m [LMC dm]

This instruction forms the logical difference of the content of the A-register and an 18-bit operand consisting of the d- and m-fields. The result remains in A. The d-field forms the highest-order six bits, and the m-field completes the lowest order 12 bits of the 18-bit operand. The logical difference forms according to the truth tables given for the 0011 instruction.

Logical difference (d) 0033 d [LMD d]

This instruction forms, in the A-register, the logical difference of bits 52-63 of the content of location d, considered as a 12-bit positive quantity and the original content of A. The highest-order six bits of A are not affected by this operation. The logical difference forms according to the truth tables given for the 0011 instruction.

Logical difference (d) long 1033 d [LMDL d]

This instruction forms, in the A-register, the logical difference of the content of location d, considered as a 16-bit positive quantity and the original content of A. Bits 46-47 of A are not affected by this operation. The logical difference forms according to the truth tables given for the 0011 instruction.

Logical difference ((d)) 0043 d [LMI d]

This instruction forms the logical difference of bits 52-63 of an operand read from storage and the original content of the A-register. Bits 46-51 of A are not affected by this operation. The storage address for the operand is in location d. The logical difference forms according to the truth tables given for the 0011 instruction.

Logical difference ((d)) long 1043 d [LMIL d]

This instruction forms the logical difference of an operand read from storage and the original content of the A-register. Bits 46-47 of A are not affected by this operation. The storage address for the operand is in location d. The logical difference forms according to the truth tables given for the 0011 instruction.

Logical difference (m+(d)) 0053 d m [LMM m,d]

This instruction forms the logical difference of bits 52-63 of an operand read from storage and the original content of the A-register. Bits 46-51 of A are not affected by this operation. The address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the address of the operand is given by m. The logical difference forms according to the truth tables given for the 0011 instruction. (See section 5.2.2.2.5 for a description of address formation.)

Logical difference (m+(d)) long 1053 d m [LMML m,d]

This instruction forms the logical difference of an operand read from storage and the original content of the A-register. Bits 46-47 of A are not affected by this operation. The address for the operand is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the address of the operand is given by m. The logical difference forms according to the truth tables given for the 0011 instruction. (See section 5.2.2.2.5 for a description of address formation.)

5.2.2.8 Replace

The replace instructions are similar to the arithmetic instructions in that they perform integer arithmetic with the contents of A as one operand and another operand whose location is specified by the instruction. The result replaces the original contents of A and the contents of the location of the other operand. The operands are considered as ones complement integers and the arithmetic is performed in ones complement.

Replace add (d) 0035 d [RAD d]

This instruction adds bits 52-63 of the content of location d, considered as a 12-bit positive integer, to the current content of the A-register. The result remains in A and also stores in location d. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the content of location d by adding six higher-order zero bits. The 16-bit result stored in location d is the lowest-order 12 bits of the resulting 18-bit sum with four higher order zero bits added. Hence, the value in A is not necessarily equal to the quantity returned to storage.

Replace add (d) long 1035 d [RADL d]

This instruction adds the content of location d, considered as a 16-bit positive integer, to the current content of the A-register. The result remains in A and also stores in location d. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the content of location d by adding two higher-order zero bits. The result stored in location d is the lowest-order 16 bits of the resulting 18-bit sum. Hence, the value in A is not necessarily equal to the quantity returned to storage.

Replace Add One (d) 0036 d [AOD d]

This instruction increases the content of bits 52-63 of location d by one count. Conceptually a value of plus one is entered into the A-register. Bits 52-63 are then read from storage and added to the A-register in an 18-bit ones complement mode. Bits 52-63 of location d are considered as a 12-bit positive integer. An 18-bit operand forms from this quantity by adding six higher-order zero bits. The result remains in A. Bits 52-63 of the resulting sum in the A-register have four higher-order zero bits added and are stored in location d. Hence, the result in A is not necessarily equal to the quantity in location d.

Replace Add One (d) long 1036 d [AODL d]

This instruction increases the content of location d by one count. Conceptually, a value of plus one is entered into the A-register. The content of location d is then read from storage and added to the A-register in an 18-bit ones complement mode. The content of location d is considered as a 16-bit positive quantity. An 18-bit operand forms from this quantity by adding two higher-order zero bits. The result remains in A. Bits 48-63 of the resulting sum in the A-register are stored in location d. Hence, the result in A is not necessarily equal to the quantity in location d.

Replace Subtract One (d) 0037 d [SOD d]

This instruction decreases the content of bits 52-63 of location d by one count. Conceptually, a value of minus one is entered into the A-register. Bits 52-63 of the content of location d are then read from storage and added to the A-register in an 18-bit ones complement mode. Bits 52-63 of location d are considered as a 12-bit positive integer. An 18-bit operand forms from this quantity by adding six higher-order zero bits. The result remains in A. Bits 52-63 of the resulting sum in the A-register have four higher-order zero bits added and are stored in location d. Hence, the result in A is not necessarily equal to the quantity in location d.

Replace Subtract One (d) long 1037 d [SODL d]

This instruction decreases the content of location d by one count. Conceptually, a value of minus one is entered into the A-register. The content of location d is then read from storage and added to the A-register in an 18-bit ones complement mode. The content of location d is considered as a 16-bit positive integer. An 18-bit operand forms from this quantity by adding two higher-order zero bits. The result remains in A. Bits 48-63 of the resulting sum in the A-register are stored in location d. Hence, the result in A is not necessarily equal to the quantity in location d.

Replace add ((d)) 0045 d [RAI d]

This instruction reads bits 52-63 of an operand from storage and adds it to the current content of the A-register. The result remains in A and is also stored in the same memory location from which the operand was read. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the 12-bit storage operand by adding six higher-order bits. Bits 52-63 of the resulting sum in the A-register have four higher-order zero bits added and returned to storage. Hence, the result in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is in location d.

Replace add ((d)) long 1045 d [RAIL d]

This instruction reads an operand from storage and adds it to the current content of the A-register. The result remains in A and is also stored in the same memory location from which the operand was read. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the 16-bit storage operand by adding two higher-order zero bits. Bits 48-63 of the resulting sum in the A-register and returned to storage. Hence, the result in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is in location d.

Replace Add One ((d)) 0046 d [AOI d]

This instruction increases bits 52-63 of an operand in storage by one count. Conceptually, a value of plus one is entered into the A-register. Bits 52-63 of the operand in storage are then read and added to the content of A in an 18-bit ones complement mode. The operand is treated as a 12-bit positive integer in this process. An 18-bit operand forms from the 12-bit storage operand by adding six higher-order zero bits. The result remains in A, and bits 52-63 have four higher-order zero bits added and are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is in location d.

Replace Add One ((d)) long 1046 d [AOIL d]

This instruction increases the value of an operand in storage by one count. Conceptually, a value of plus one is entered into the A-register. The storage operand is then read and added to the content of A in an 18-bit ones complement mode. The operand is treated as a 16-bit positive integer in this process. An 18-bit operand forms from the 16-bit storge operand by adding two higher-order zero bits. The result remains in A, and bits 48-63 are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is in location d.

Replace Subtract One ((d)) 0047 d [SOI d]

This instruction reads bits 52-63 of an operand from storage, decreases its value by one count and returns bits 52-63 of the result to the same storage location. Conceptually, a value of minus one is entered into the A-register. Bits 52-63 of the operand in storage are then read and added to the content of A in an 18-bit ones complement mode. The operand is treated as a 12-bit positive integer in this process. An 18-bit operand forms from the 12-bit storage operand by adding six higher-order zero bits. The result remains in A, and bits 52-63 have four higher-order zero bits added and are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is in location d.

Replace Subtract One ((d)) long 1047 d [SOIL d]

This instruction reads an operand from storage, decreases its value by one count, and returns the result to the same storage location. Conceptually, a value of minus one is entered into the A-register. The operand then reads from storage and adds to the content of A in an 18-bit ones complement mode. The operand is treated as a 16-bit positive integer in this process. An 18-bit operand forms from the 16-bit storage operand by adding two higher-order zero bits. The result remains in A, and bits 48-63 are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is in location d.

Replace add (m+(d)) 0055 d m [RAM m,d]

This instruction reads bits 52-63 of an operand from storage and adds it to the current content of the A-register. The result remains in A, and bits 52-63 store in the same memory location from which the operand was read. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the 12-bit storage operand by adding six higher-order zero bits. Bits 52-63 of the result in A, have four higher-order zero bits added and are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given in m. (See section 5.2.2.2.5 for a description of address formation.)

Replace add (m+(d)) long 1055 d m [RAML m,d]

This instruction reads an operand from storage and adds it to the current content of the A-register. The result remains in A, and bits 48-63 store in the same memory location from which the operand was read. The addition is in an 18-bit ones complement mode. An 18-bit operand forms from the 16-bit storage operand by adding two higher-order zero bits. Bits 48-63 of the result in A are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Replace Add One (m+(d)) 0056 d m [AOM m,d]

This instruction reads bits 52-63 of an operand from storage, increases its value by one count, and returns bits 52-63 of the result to the same storage location. Conceptually, a value of plus one is entered into the A-register. The operand then reads from storage and adds to the content of A in an 18-bit ones complement mode. The operand is treated as a 12-bit positive integer in this process. An 18-bit operand forms from the 12-bit storage operand by adding six higher-order zero bits. The result remains in A, and bits 52-63 have four higher-order zero bits added and are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Replace Add One (m+(d)) long 1056 d m [AOML m,d]

This instruction reads an operand from storage, increases its value by one count, and returns the result to the same storage location. Conceptually, a value of plus one is entered into the A-register. The operand then reads from storage and adds to the content of A in an 18-bit ones complement mode. The operand is treated as a 16-bit positive integer in this process. An 18-bit operand forms from the 16-bit storage operand by adding two higher-order bits. The result remains in A, and bits 48-63 are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Replace Subtract one (m+(d)) 0057 d m [SOM m,d]

This instruction reads bits 52-63 of an operand from storage, decreases its value by one count, and returns bits 52-63 of the result to the same storage location. Conceptually, a value of minus one is entered into the A-register. The operand then reads from storage and adds to the content of A in an 18-bit ones complement mode. The operand is treated as a 12-bit positive integer in this process. An 18-bit operand forms from the 12-bit storage operand by adding six higher-order zero bits. The result remains in A, and bits 52-63 have four higher-order zero bits added and are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Replace Subtract one (m+(d)) long 1057 d m [SOML m,d]

This instruction reads an operand from storage, decreases its value by one count, and returns the result to the same storage location. Conceptually, a value of minus one is entered into the A-register. The operand then reads from storage and adds to the content of A in an 18-bit ones complement mode. The operand is treated as a 16-bit positive integer in this process. An 18-bit operand forms from the 16-bit storage operand by adding two higher-order bits. The result remains in A, and bits 48-63 are returned to storage. Hence, the value in A is not necessarily equal to the quantity returned to storage. The storage address for reading the operand and storing the result is formed by adding the m-field to the contents of location d in ones complement mode. If the d-field is zero, then the storage address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

5.2.2.9 Branch

The branch instructions provide the capability to depart from the sequential execution of instructions.

Unconditional jump d   0003 d [UJN d]

This instruction causes a branch to a location forward or backward as specified by d. The d-field is considered as a 6-bit ones complement number. If d is in the range 0-31, the branch is forward d locations. If d is in the range 32-63, the branch is backward 63−d locations.

Zero jump d   0004 d [ZJN d]

If (A) is zero (all bits of A are clear), then this causes a branch to a location forward or backward as specified by d. The d-field is considered as a 6-bit ones complement number. If d is in the range 0-31, the branch is forward d locations. If d is in the range 32-63, the branch is backward 63−d locations.

Nonzero jump d   0005 d [NJN d]

If (A) is non-zero (all bits of A are not clear), then this causes a branch to a location forward or backward as specified by d. The d-field is considered as a 6-bit ones complement number. If d is in the range of 0-31, the branch is forward d locations. If d is in the range 32-63, the branch is backward 63−d locations.

Plus jump d   0006 d [PJN d]

If (A) is positive (bit 46 of A is clear), then this causes a branch to a location forward or backward as specified by d. The d-field is considered as a 6-bit ones complement number. If d is in the range 0-31, the branch is forward d locations. If d is in the range 32-63, the branch is backward 63−d locations.

Minus jump d   0007 d [MJN d]

If (A) is negative (bit 46 of A is set), then this causes a branch to a location forward or backward as specified by d. The d-field is considered as a 6-bit ones complement number. If d is in the range 0-31, the branch is forward d locations. If d is in the range 32-63, the branch is backward 63−d locations.

Long jump to m+(d)   0001 d m [LJM m,d]

This instruction branches to an address formed from m+(d). The m-field is added to the contents of location d in a ones complement mode. The result forms an address which is used to obtain the first word of the new instruction sequence. If the d-field is zero, then the address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

Return jump to m+(d)   0002 d m [RJM m,d]

This instruction stores the current program address plus two ((P)+2) in the address formed from m+(d). The instruction then branches to m+(d)+1. To form the storage address, the m-field is added to of the contents of location d in a ones complement mode. If the d-field is zero, then the address is given by m. (See section 5.2.2.2.5 for a description of address formation.)

5.2.2.10 Central Memory Access

The central memory access instructions provide the capability to read and write central memory words to and from the PP memory. The PPs have read access to all central memory storage locations, *while write and exchange accesses are controlled by the OS Bounds Register (except for I0).* Central memory addressing is performed with real rather than virtual memory addresses.

5.2.2.10.1 R Register

The central memory address is formed from the contents of the R-register and the A-register. If bit 46 of the A-register is a one or the Relocation flag is set (I0 only) during a PP central memory read/write instruction, then the contents of the R-register are added to the contents of bits 47-63 of the A-register to form the central memory address. If bit 46 of the A-register and the Relocation flag are zero during a central memory read/write instruction, then no address relocation is performed.

The R-register contains an absolute 64-word starting boundary within central memory. When relocation is desired, an absolute central memory address is formed by concatenating six zeroes to the rightmost end of the contents of the R-register and then adding bits 47-63 of (A). See Figures 5.2-1 and 5.2-2.

5.2.2.10.2 Relocation Flag (I0 only)

The I0 contains a Relocation flag in each PP, except for those described in 5.2.2.10.3. When it is set, it will force (R)+(A) addition to occur on every central memory reference. Its purpose is to force relocated central memory references without having to set bit 17 of the A-register. This flag is cleared by deadstart and execution of the 0024 Load R-register intruction and is set by execution of the 1011, 1012 and 1013 Load R-register Upper instructions.

This page has been left blank intentionally.

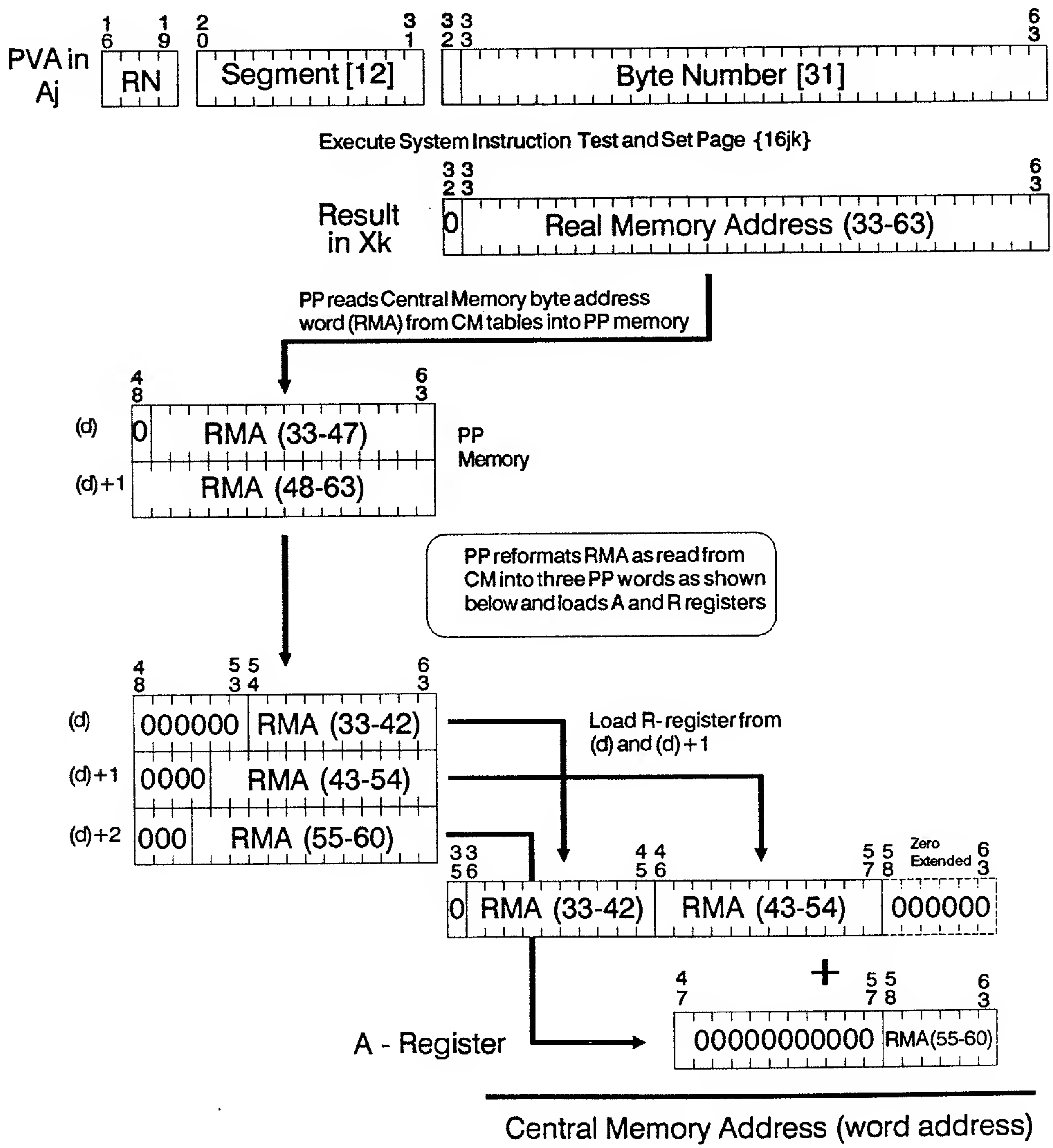


Figure 5.2-1. CYBER 170 Mode R-Register (I1, I2, I4 & I0)

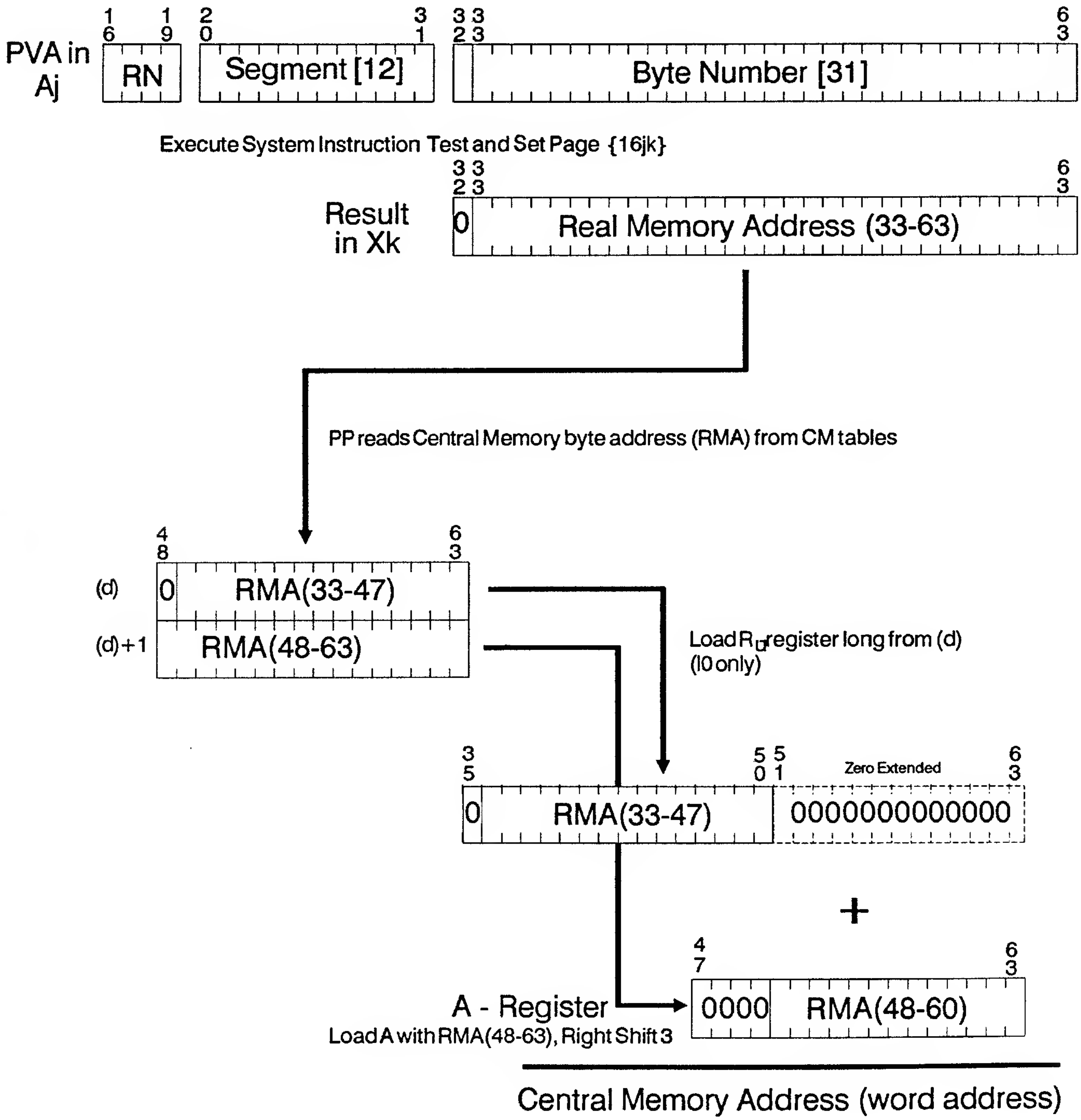


Figure 5.2-2. CYBER 180 Mode R-Register (I0 only)

Central Memory Write Access

The OS Bounds Maintenance Register divides central memory into an upper and lower region for dual-state operation. A bit in the OS Bounds Register for each PP indicates to which region central memory writes and exchanges are allowed. A set bit indicates the lower region: PP CM address < OS Boundary. While a cleared bit indicates the upper region: OS Boundary ≤ PP CM address. If the PP attempts to access its prohibited region when the Enable OS Bounds Checking bit is set in the Environment Control register the write or exchange will not occur, the OS Bounds Fault will be set in a maintenance register and if the Enable Error Stop is set in the Environment Control register the PP will be idled.

The IOU instruction for which CM addresses are verified are:

0026 Exchange Jump
0062 Central Write
1062 Central Write
0063 Central Write
1063 Central Write
1000 Central Read and Set Lock
1001 Central Read and Clear Lock
1070 Central DMA Write

NOTE: The OS Bounds Register is not implemented in the I0.

Load R-register 0024 d [LRD d]

This instruction loads the 22-bit R-register from (d) and (d)+1. Provided d is non-zero, bits 46-57 of R are loaded from bits 52-63 of (d)+1, the remaining bits 36-45 are loaded from bits 54-63 of (d). If the d-field is zero, then the instruction is a Pass. The PP's R-flag is cleared

Store R-register 0025 d [SRD d]

This instruction stores the contents of the 22-bit R-register into locations (d) and (d)+1. Provided d is non-zero, bits 46-57 of R are stored into (d)+1, and the remaining bits 36-45 of R are stored into bits 54-63 of (d). Bits 48-53 of (d) and bits 48-51 of (d)+1 are cleared. If the d-field is zero, then the instruction is a Pass.

Load R-register Long 1011 d [LRDL d] **(IO only)**

This instruction loads a 16 bit field (bits 35-50) within the R-register from (d). The field is defined by its use in the central memory address generation scheme described in section 5.2.2.10.2. The field is loaded from bits 48-63 of (d). The PP's R-flag is set. If the R-register extends beyond the 16 bit field defined above, the state of the other part(s) of the R-register is undefined after execution of this instruction.

Store R-register Long 1014 d [SRDL d] **(IO only)**

This instruction stores the contents of a 16 bit field (bits 35-50) within the R-register into (d). The field is defined by its use in the central memory address generation scheme described in section 5.2.2.10.2. The field is stored in bits 48-63 of (d).

Load R-register Long 1012 d [LRIL d] **(IO only)**

This instruction loads a 16 bit field (bits 35-50) within the R-register from ((d)). The field is defined by its use in the central memory address generation scheme described in section 5.2.2.10.2. The field is loaded from bits 48-63 of ((d)). The PP's R-flag is set. If the R-register extends beyond the 16 bit field defined above, the state of the other part(s) of the R-register is undefined after execution of this instruction.

Store R-register Long 1015 d [SRIL d] **(IO only)**

This instruction stores the contents of a 16 bit field (bits 35-50) within the R-register into ((d)). The field is defined by its use in the central memory address generation scheme described in section 5.2.2.10.2. The field is stored in bits 48-63 of ((d)).

Load R-register Long 1013 d m [LRML m,d] **(IO only)**

This instruction loads a 16 bit field (bits 35-50) within the R-register from bits 47-63 of (m+(d)) i.e. an address formed by adding bits 48-63 of (m) to bits 48-63 of (d) in a 16-bit ones complement mode. The R-register field is defined by its use in the central memory address generation scheme described in section 5.2.2.10.2. If the d-field is zero then the memory address is given by (m). The PP's R-flag is set. If the R-register extends beyond the 16 bit field defined above, the state of the other part(s) of the R-register is undefined after execution of this instruction.

Store R-register Long 1016 d m [SRML m,d] **(IO only)**

This instruction stores a 16 bit field (bits 35-50) within the R-register into bits 48-63 of ((m+d)) i.e. an address formed by adding bits 48-63 of (m) to bits 48-63 of (d) in a 16-bit ones complement mode. The R register field is defined by its use in the central memory address generation scheme described in section 5.2.2.10.2. If the d-field is zero then the memory address is given by (m).

Central read from (A) to d 0060 d [CRD d] ***(Not on I0)***

This instruction transfers bits 4-63 of one central memory to bits 52-63 of five consecutive PP memory words. Bits 0-3 of the central memory word are discarded and the remaining 60 bits are disassembled from the left into five 12-bit bytes. This unpacking is illustrated below. The address of the central memory word is specified by (R)+(A). The address of the first PP memory word is specified by d.

Central memory word

0	4	16	28	40	52 – 63
(4)	a(12)	b(12)	c(12)	d(12)	e(12)

PP memory words

	48	52 – 63
d	0(4)	a(12)
d+1	0(4)	b(12)
d+2	0(4)	c(12)
d+3	0(4)	d(12)
d+4	0(4)	e(12)

Central read from (A) to d long   1060 d [CRDL d]

This instruction transfers one central memory word to four consecutive PP memory words. The central memory word is disassembled from the left. This unpacking is illustrated below. The address of the central memory word is specified by (R)+(A). The address of the first PP word is specified by d.

Central memory word

0	16	32	48	63
a(16)	b(16)	c(16)	d(16)	

PP memory words

	48 – 63
d	a(16)
d+1	b(16)
d+2	c(16)
d+3	d(16)

Central read (d) words from (A) to m 0061 d m [CRM m,d] ***(Not on I0)***

This instruction transfers bits 4-63 of consecutive central memory words to bits 52-63 of consecutive PP memory words. Bits 0-4 of each central memory word are discarded and the remaining 60 bits disassembled from the left into five 12-bit bytes. See 060 instruction for illustration of unpacking. The address of the first central memory word is specified by (R)+(A). The address of the first PP word is specified by m. The number of central memory words transferred is specified by (d). Upon completion, A contains the non-relocated portion of the central memory address plus one of the last central memory word transferred. Note that if the value of bits 47-63 of A exceeds $(2^{17})-1$, then bit 46 of A will be toggled. This could cause the operation to switch between direct address and relocation address modes. Note also that if the last word transferred is from a relative address of 377776_8 and relocation is in affect, then the A-Register will be cleared and the value returned in A may not point to the last word transferred plus one.

Central read (d) words from (A) to m long 1061 d m [CRML m,d]

This instruction transfers consecutive central memory words to consecutive PP memory words. Each central memory word is disassembled from the left. See the 1060 instruction for illustration of this unpacking. The address of the first central memory word is specified by (R)+(A). The address of the first PP word is specified by m. The number of central memory words transferred is specified by (d). Upon completion, A contains the non-relocated portion of the central memory address plus one of the last central memory word transferred. Note that if the value of bits 47-63 of A exceeds $(2^{17})-1$, then bit 46 of A will be toggled. This may cause the operation to switch between direct address and relocation address modes. Note also that if the last word transferred is from a relative address of 377776_8 and relocation is in affect, then the A-Register will be cleared and the value returned in A may not point to the last word transferred plus one.

Central write from d to (A) 0062 d [CWD d] ***(Not on I0)***

This instruction transfers bits 52-63 of five consecutive PP memory words (bits 0-4 of the words are ignored) to bits 4-63 of one central memory word (bits 0-3 are cleared). These bytes are assembled from the left as illustrated below. The address of the central memory word is specified by (R)+(A) and is verified against the OS Bounds Register. The address of the first PP word is specified by d.

On the I2, a cache invalidation signal is sent to central memory with every word written. On the I4, a cache invalidation signal is sent to central memory with every word written if the OS Bounds bit is set.

PP memory words

	48	52 ... 63
d	(4)	a(12)
d+1	(4)	b(12)
d+2	(4)	c(12)
d+3	(4)	d(12)
d+4	(4)	e(12)

Central memory word

0	4	16	28	40	52 ... 63
(4)	a(12)	b(12)	c(12)	d(12)	e(12)

Central write from d to (A) long   1062 d [CWDL d]

This instruction transfers four consecutive PP memory words to one central memory word. This packing is illustrated below. The address of the first PP word is specified by d. The address of the central memory word is specified by (R)+(A) *and is verified against the OS Bounds Register.*

On the I4, a cache invalidation signal is sent to central memory with every word written if the OS Bounds bit is set.

PP memory words

	48 … 63
d	a(16)
d+1	b(16)
d+2	c(16)
d+3	d(16)

Central memory word

0	16	32	48 … 63
a(16)	b(16)	c(16)	d(16)

Central write (d) words from m to (A) 0063 d m [CWM m,d] ***(Not on I0)***

This instruction transfers bits 52-63 of consecutive PP memory words to bits 4-63 of consecutive central memory words. See the 0062 instruction for illustration of this packing. The address of the first PP memory word is specified by m. The address of the first central memory word is specified by (R)+(A). The number of central memory words transferred is specified by (d). Each central memory address generated is verified against the OS Bounds Register. Upon completion, A contains the non-relocated portion of the central memory address plus one of the last central memory word transferred. Note that if the value of bits 47-63 of A exceeds $(2^{17})-1$, then bit 46 of A will be toggled. This could cause the operation to switch between direct address and relocation address modes. Note also that if the last word transferred is from a relative address of 377776_8 and relocation is in affect, then the A-Register will be cleared and the value returned in A may not point to the last word transferred plus one.

On the I2, a cache invalidation signal is sent to central memory each time the address modulo 4 is equal to three and when the last word of the transfer is written. On the I4, if the OS Bounds bit is set, a cache invalidation signal is sent to central memory each time the address modulo 4 is equal to three and when the last word of the transfer is written.

Central write (d) words from m to (A) long 1063 d m [CWML m, d]

This instruction transfers consecutive PP memory words to consecutive central memory words. Four PP words are packed from the left into each central memory word. The address of the first PP memory word is specified by m. The address of the first central memory word is specified by (R)+(A). The number of central memory words transferred is specified by (d). *Each central memory address generated is verified against the OS Bounds Register.* Upon completion, A contains the non-relocated portion of the central memory address plus one of the last central memory word transferred. Note that if the value of bits 47-63 of A exceeds $(2^{17})-1$, then bit 46 of A will be toggled. This may cause the operation to switch between direct address and relocation address modes. Note also that if the last word transferred is from a relative address of 377776_8 and relocation is in affect, then the A-Register will be cleared and the value returned in A may not point to the last word transferred plus one.

On the I4, if the OS Bounds bit is set, a cache invalidation signal is sent to central memory each time the address modulo 4 is equal to three and when the last word of the transfer is written.

Central read and set lock from d to (A) 1000 d [RDSL d]

This instruction performs a logical "OR" function between four consecutive PP memory words and one central memory word with the result replacing the central memory word. The original contents of the central memory word replaces the four PP memory words. See the 1060 and 1062 instructions for the packing and unpacking involved. The address of the first PP word is specified by d. The address of the central memory word is specified by (R)+(A) *and is verified against the OS Bounds Register.*

A serialization function is performed before this instruction begins and again at its ending. Execution of this instruction is delayed until all previous accesses to central memory by the IOU are completed. No other accesses from any port shall be permitted to the central memory word from the beginning of the read to the end of the write. Execution of subsequent instructions by the IOU are delayed until all central memory accesses from this instruction are completed.

Central read and clear lock from d to (A) 1001 d [RDCL d]

This instruction performs a logical "AND" function between four consecutive PP memory words and one central memory word with the result replacing the central memory word. The original contents of the central memory word replaces the four PP memory words. See the 1060 and 1062 instructions for the packing and unpacking involved. The address of the first PP word is specified by d. The address of the central memory word is specified by (R)+(A) *and is verified against the OS Bounds Register.*

A serialization function is performed before this instruction begins and again at its ending. Execution of this instruction is delayed until all previous accesses to central memory by the IOU are completed. No other accesses from any port shall be permitted to the central memory word from the beginning of the read to the end of the write. Execution of subsequent instructions by the IOU are delayed until all central memory accesses from this instruction are completed.

5.2.2.11 Input/Output

These instructions select an external device and transfer data to or from that device. The instructions also determine whether a channel or external device is available and ready to transfer data. The preparatory steps ensure that the data transfer is carried out in an orderly fashion.

Each external device has a set of external function codes that the PP uses to establish modes of operation and to start and stop data transfer. The devices are also capable of detecting certain errors, which they indicate to the controlling PP.

Jump to m if channel c active 00640 c m [AJM m,c]

This instruction branches to the location specified by m if channel c is active.

Test and Set channel c flag 00641 c m [SCF m,c]

This instruction branches to the location specified by m if the channel c flag is set, otherwise it sets the channel flag and exits. One may unconditionally set the channel flag by setting m to P+2.

Note: A conflict condition can occur when two or more PPs are trying to execute a 00641 instruction on the same channel simultaneously. Only on the maintenance channel (Channel 17) will this be resolved by letting the PP in the lowest physical barrel see the true status of the flag and to the other PP's in conflict the flag shall appear as set [and hence take a jump].

Jump to m if channel c flag set 1064X c m [FSJM m,c]

This instruction branches to the location specified by m if the flag for channel c is set.

Jump to m if channel c inactive 00650 c m [IJM m,c]

This instruction branches to the location specified by m if channel c is inactive.

Clear channel c flag 00651 c m [CCF c]

This instruction clears the flag in the channel specified by c. The m-field is required but not used.

Jump to m if channel c flag clear 1065X c m [FCJM m,c]

This instruction branches to the location specified by m if the flag for channel c is clear.

Jump to m if channel c full 00660 c m [FJM m,c]

This instruction branches to the location specified by m if channel c is full.

Test and clear channel c error flag set 00661 c m [SFM m,c]

This jump instruction branches to the location specified by m if the channel c error flag is set, and clears the error flag.

Jump to m on Uncorrected Error 1066 m [VEJM m] **(I0 only)**

This instruction branches to the location specified by m if the Uncorrected Error indication specifically of the PP that is executing this instruction is set. The d field is not significant. Otherwise it passes to the next instruction. The circumstances under which the Uncorrected error indication is set are model dependent and can be found in the appropriate Engineering specification.

Jump to m if channel c empty 00670 c m [EJM m,c]

This instruction branches to the location specified by m if channel c is empty.

Test and clear channel c error flag clear 00671 c m [CFM m,c]

This jump instruction branches to the location specified by m if the channel c error flag is clear, else it clears the error flag.

Input to A from channel c when active 00700 c [IAN c]

This instruction transfers A word from channel c to 16 bits 48-63 of A. Bits 46-47 of A are cleared. The instruction waits for the channel to become active and full before executing.

Note: If a 12-bit external interface is used on the channel, bits 46-51 of A will be zero. If an 8-bit external interface is used on the channel, then bits 46-55 of A will be zero.

Input to A from channel c if active 00701 c [IAN 40B+c]

This instruction transfers a word from channel c to bits 48-63 of A. Bits 46,47 of A are cleared. If the channel is inactive or becomes inactive before becoming full, then no transfer takes place and the instruction exits with (A)=0.

Note: If a 12-bit external interface is used on the channel, then bits 46-51 of A will be zero. If an 8-bit external interface is used on the channel, then bits 46-55 of A will be zero.

Input (m) channel words to 64-bit central memory (R)+(A) (I0 only)
from 16-bit channel c 1070X c m [CHCM m, c]

This instruction transfers bits 48-63 of successive (16-bit) channel words to packed 64-bit central memory words. During the transfer four 16-bit channel words from channel c are packed into one 64-bit central memory word as illustrated below. (R) will be added to bits 0-16 of (A) for every central memory reference if the relocation flag is set or bit 17 of (A) is set, otherwise central memory address references will be specified by (A) alone. The address for the operand (m) specifying the number of 16-bit words to be transferred is given by bits 48-63 of the m-field. As the transfer proceeds a representation of the number of words remaining is decremented for each channel word transferred and (A) is incremented for each central memory word transferred. It is important that (A) not be initialized such as to allow it to increment past a value of 777776_8 since it will then advance to a value of 000000_8 and (R) will not be advanced.

The transfer is complete when either all words are transferred or the channel becomes inactive. On termination the last CM word accessed will be left justified and zero filled. Upon completion, (R)+(A) contains the central memory address plus one of the last central memory word transferred if the relocation flag is set or if bit 17 of (A) is set otherwise (A) contains the central memory address plus one and (m) contains the number of channel words not transferred. If the instruction is executed with the channel initially inactive, no transfer takes place and the instruction exits with (A) and (m) unchanged.

The maximum transfer size is 64K channel words (128K bytes).

I/O Data Channel Word

48 — 63
(16) a
(16) b
(16) c
(16) d

Central Memory Word

00 — 15	16 — 31	32 — 47	48 — 63
(16) a	(16) b	(16) c	(16) d

Input (A) words to m from channel c   0071X c m [IAM m,c]

This instruction transfers successive words from channel c to consecutive PP memory words. The address of the first PP memory word is specified by m. The number of words to be transferred is specified by (A). The transfer is complete when either (A)=0 or the channel becomes inactive. If the termination is caused by an inactive channel, the next PP memory word is cleared, and (A) contains the difference of its initial value and the number of words actually transferred from the channel.

If the instruction is executed with the channel initially inactive, no transfer takes place and the instruction exits with (A) unchanged and the PP memory word specified by m is set to 0.

Note: If a 12-bit external interface is used on the channel, bits 48-51 of PP memory will be zero. If an 8-bit external interface is used on the channel, then bits 48-55 of PP memory will be zero.

Input A words to m from channel c packed 1071X c m [IAPM m,c] (Not on I0)

This instruction transfers bits 52-63 of successive words from channel c to consecutive PP memory words. During this transfer, processing occurs such that four 12-bit channel words (48 bits) are packed into three 16-bit PP memory words, as illustrated below. Bits 48-51 of the 16-bit channel words are ignored. The address of the first PP memory word is specified by m. The number of channel words to be transferred is specified by (A). The transfer is complete when either (A)=0 or the channel becomes inactive. If the termination is caused by an an end of word count (A=0), and the number of channel words transferred is not a multiple of four, then the last PP memory word will be zero filled. If the termination is caused by an inactive channel, then PP memory words will be zero filled up to the next four channel word boundary. For example, if 17 channel words were transmitted followed by an inactive, then the first 16 channel words will be stored in the first 12 PP memory words (starting address m), the thirteenth PP memory word will contain the seventeenth channel word in bits 48-59, and bits 60-63, together with the next two PP memory words will be zeroed.

If the instruction is executed with the channel initially inactive, no transfer takes place and the instruction exits with A unchanged, and the next three PP memory words specified by m, m+1, and m+2 are set to 0.

This instruction allows 16-bit PP memory words to be read from 12-bit external devices.

Channel words

48	52		63
(4)	a(12)		
(4)	b(4)	c(8)	
(4)	d(8)		e(4)
(4)	f(12)		

PP memory words

	48	52	56	60 63
m	a(12)			b(4)
m+1	c(8)		d(8)	
m+2	e(4)	f(12)		

Output from A to channel c when active 00720 c [OAN c]

This instruction transfers bits 48-63 of (A) to channel c. The instruction waits for the channel to become active and empty before executing. The content of A is not altered.

Note: If a 12-bit external interface is used on the channel, bits 48-51 of the channel word are not transmitted to the external device and are lost. If an 8-bit external interface is used on the channel, then bits 48-55 of the channel word are not transmitted to the external device and are lost.

Output from A to channel c if active 00721 c [OAN 40B+c]

This instruction transfers bits 48-63 of (A) to channel c. If the channel is inactive, then no transfer takes place and the instruction exits. The content of A is not altered.

Note: If a 12-bit external interface is used on the channel, then bits 48-51 of the channel word are not transmitted to the external device and are lost. If an 8-bit external interface is used on the channel, then bits 48-55 of the channel word are not transmitted to the external device and are lost.

Output (m) channel words from 64-bit central memory (R)+(A) to 16-bit channel c 1072X c m [CMCH m, c] **(IO only)**

This instruction transfers packed 64-bit central memory words into successive 16-bit I/O data channel words. During the transfer, a 64-bit central memory word is unpacked into four 16-bit I/O data channel words. (R) will be added to bits 0-16 of (A) for every central memory reference if the relocation flag is set or bit 17 of (A) is set, otherwise central memory address references will be specified by (A) alone. The address for the operand (m) specifying the number of 16-bit words to be transferred is given by bits 48-63 of the m-field. As the transfer proceeds a representation of the number of words remaining is decremented for each channel word transferred and (A) is incremented for each central memory word transferred. It is important that (A) not be initialized such as to allow it to increment past a value of 777776_8 since it will then advance to a value of 000000_8 and (R) will not be advanced.

The transfer is complete when either all words are transferred or the channel become inactive. Upon completion, (R)+(A) contains the central memory address plus one of the last central memory word transfered if the relocation flag is set or if bit 17 of (A) is set otherwise (A) contains the central memory address plus one and (m) contains the number of channel words not transfered. If the instruction is executed with the channel initially inactive, no transfer takes place and the instruction exits with (A) and (m) unchanged.

The maximum transfer size is 64K channel words (128K bytes).

Output A words from m on channel c 0073X c m [OAM m,c]

This instruction transfers the contents of consecutive PP memory words as successive words on channel c. The address of the first PP memory word is specified by m. The number of words to be transferred is specified by (A). The transfer is complete when either (A)=0 or the channel becomes inactive. If the termination is caused by an inactive channel then (A) contains the difference of its initial value and the number of words actually transferred on the channel.

If the instruction is executed with the channel initially inactive, no transfer takes place and the instruction exits with (A) unchanged.

Note: If a 12-bit external interface is used on the channel, then bits 48-51 of the channel word are not transmitted to the external device and are lost. If an 8-bit external interface is used on the channel, then bits 48-55 of the channel word are not transmitted to the external device and are lost.

Output A words from m on channel c packed 1073X c m [OAPM m,c] (Not on I0)

This instruction transfers consecutive PP memory words as bits 52-63 of successive words on channel c. During the transfer, processing occurs such that the contents of three 16-bit PP memory words result in four 12-bit channel words. Bits 48-51 of the 16-bit channel words are cleared. This packing is illustrated in the 1071 instruction. The address of the first PP word is specified by m. The number of channel words to be transferred is specified by (A). The transfer is complete when either (A)=0 or the channel becomes inactive. If the termination is caused by an inactive channel then (A) contains the difference of its initial value and the number of words actually transferred on the channel.

If the instruction is executed with the channel initially inactive, no transfer takes place and the instruction exits with (A) unchanged.

Activate channel c 00740 c [ACN c]

This instruction prepares channel c for I/O transfer operations by setting the channel active. If the channel is initially active, then the instruction will wait for the channel to become inactive before executing.

Unconditionally activate channel c 00741 c [ACN 40B+c]

This instruction prepares channel c for I/O transfer operations by setting the channel active. The instruction will execute regardless of the active/inactive status of the channel.

Master clear channel c 1074 c [MCLR c] (I0 only)

This instruction unconditionally Master Clears I/O channel c. After the execution of this instruction the channel is active and empty. Note that the clearing process in the channel takes one microsecond, and thus the channel must not be accessed from a PP during this period.

Deactivate channel c 00750 c [DCN c]

This instruction terminates I/O operations on channel c by setting the channel inactive. If the channel is initially inactive, then the instruction will wait for the channel to become active before executing.

If the instruction is executed after an output instruction without waiting for the channel to become empty, then the last channel word transferred may be lost.

Unconditionally deactivate channel c 00751 c [DCN 40B+c]

This instruction terminates I/O operations on channel c by by setting the channel inactive. The instruction will execute regardless of the active/inactive status of the channel.

If this instruction is executed after an output instruction without waiting for the channel to become empty, the last channel word transferred may be lost.

Function A on channel c when inactive 00760 c [FAN c]

This instruction transfers bits 48-63 of (A) to channel c as a function code. If the channel is initially active, the instruction will wait for the channel to become inactive before executing. The contents of A is not altered.

Note: If a 12-bit external interface is used on the channel, then bits 48-51 of A are not transmitted to the external device and are lost. If an 8-bit external interface is used on the channel, then bits 48-55 of A are not transmitted to the external device and are lost.

Function A on channel c if inactive 00761 c [FAN 40B+c]

This instruction transfers bits 48-63 of (A) to channel c as a function code. If the channel is initially active, then the function is not transferred on the channel, and the instruction exits. The content of A is not altered.

Note: If a 12-bit external interface is used on the channel, then bits 48-51 of A are not transmitted to the external device and are lost. If an 8-bit external interface is used on the channel, then bits 48-55 of A are not transmitted to the external device and are lost.

Function m on channel c when inactive 00770 c m [FNC m,c]

This instruction transfers m to channel c as a function code. If the channel is initially active, then the instruction will wait for the channel to become inactive before executing.

Note: If a 12-bit external interface is used on the channel, then bits 48-51 of the function word m are not transmitted to the external device and are lost. If an 8-bit external interface is used on the channel, then bits 48-55 of the function word m are not transmitted to the external device and are lost.

Function m on channel c if inactive 00771 c m [FNC m,40B+c]

This instruction transfers m to channel c as a function code. If the channel is initially active, then the function is not transferred on the cHannel, and the instruction exits.

Note: If a 12-bit external interface is used on the channel, then bits 48-51 of the function word m are not transmitted to the external device and are lost. If an 8-bit external interface is used on the channel, then bits 48-55 of the function word m are not transmitted to the external device and are lost.

5.2.2.12 Other

Pass 0000 d [PSN]

See Appendix E (PP instructions) for complete list of Pass instructions. The Pass instructions perform no operation.

Wait (A) 1017 [WAIT] **(I0 only)**

This instruction causes the PP to wait for (A) microseconds while A counts down every microsecond. The instruction exits with (A)=0. This provides a model-independent wait time of 1 microsecond to 256 milliseconds.

PP Keypoint 0027 [KEYP]

This instruction executes as a Pass instruction but allows sensing of its execution by external monitoring equipment through a test point.

Exchange jump 0026 d ***(Not on I0)***

This instruction provides the capability for PP programs to control the execution of the CPU in CYBER 170 state. The Exchange Request is transmitted to the CPU that has been designated as the CPU for performing CYBER 170 state execution. See Section 7.12 of this specification for further details.

If an Exchange Request for any PP is outstanding, then a further Exchange Request from any PP will cause that PP to wait until completion of the outstanding Exchange Request.

This instruction does not complete until an Exchange Accept signal is returned to the IOU by the CPU. The Exchange Accept is sent upon completion of the requested CYBER 170 exchange jump. The PP Halted bit in the IOU Status Summary register shall not be set as the result of a PP waiting for an Exhange Accept or waiting to issue an Exchange Request.

The value of d controls the action taken to process the exchange request in CYBER 170 state.

d = 0 - Unconditional exchange jump 0026 00 [EXN] ***(Not on I0)***

An exchange jump is unconditionally performed at the address specified by (R)+(A). This exchange package FWA address is verified against the OS Bounds Register and if in the prohibited region and the Enable OS Bounds Checking bit is set in the Environment Control register, the exchange will not occur, the OS Bounds Fault will be set and if the Enable Error Stop is set in the Environmental Control Register the PP will be idled. Note that only the FWA of the exchange package is verified and thus the exchange package could extend across the OS boundry.

d = 10B - Monitor exchange jump 0026 10 [MXN] ***(Not on I0)***

An exchange jump is conditionally performed at the address specified by (R)+(A). This exchange package FWA address is verified against the OS Bounds Register and if in the prohibited region and the Enable OS Bounds Checking bit is set in the Environment Control register, the exchange will not occur, the OS Bounds Fault will be set and if the Enable Error Stop is set in the Environmental Control Register the PP will be idled. Note that only the FWA is verified and an exchange package could extend across the OS boundary. Otherwise if the monitor flag is clear, the exchange jump is performed and the monitor flag is set. If the monitor flag is set, the exchange jump is not performed and this instruction becomes a Pass instruction.

d = 20B - Monitor exchange jump to MA 0026 20 [MAN] ***(Not on I0)***

An exchange jump is conditionally performed at the address specified by the CPU Monitor Address (MA) register. If the monitor flag is clear, the exchange jump is performed and the monitor flag is set. If the monitor flag is set, the exchange jump is not performed and ths instruction becomes a Pass instruction.

If d = 30B, then the instruction executes as if d = 20B.

If an exchange request for any PP is outstanding, then a further exchange request from any PP will cause that PP to wait until completion of the outstanding exchange request.

Interrupt processor 1026 d [INPN d]

This instruction transmits an interrupt signal for the CPU on the memory port specified by d. This interrupt signal is transmitted via the memory port interface provided to transmit interrupts between processors. This interrupt signal causes the External Interrupt bit to be set in the CPU Monitor Condition Register. See section 2.8.1 for definition of possible actions that may be taken by the CPU. A serialization function is performed before this instruction begins execution. That is, execution of this instruction is delayed until all previous central memory accesses on the part of the interrupting processor are complete.

Note: The I0 requires the five least significant bits of the A register to be cleared prior to executing this instruction.

5.3 I/O CHANNELS

A PP interfaces to an I/O channel to communicate with external devices or other PPs. Each I/O channel is composed of an internal interface and a modular external interface. The internal interface allows common hardware and software logic to control the external devices. The external interface allows the IOU to communicate with the external devices using a variety of channels including *6000 Series Channel,* CYBER 170 Channel, *CYBER 180 Maintenance Channel and CYBER 180 Channel.*

5.3.1 Internal Interface

The internal interface consists of bidirectional data and status registers. The data register is 17 bits long (16 data bits and 1 parity bit). The status register is 4 bits long and contains the 'active', 'full', 'flag', and 'error flag' bits.

5.3.1.1 Active bit

The active bit is set (channel active) either by a PP or (in the case of the CYBER 170 channel) an external device. This condition indicates that the channel is reserved for channel communication.

The active bit is cleared (channel inactive) either by a PP or an external device. This action denotes that the channel communication is complete.

The active bit is set from a PP by the use of the activate instruction (00740, 00741) or by a function instruction (00760, 00761, 00770, 00771). The active bit is cleared from a PP by the use of the deactivate instruction (00750, 00751). Normally, external devices only clear the active bit either at the end of an input transfer or in response to a function instruction.

The state of the active bit is sensed from a PP by the use of the active/inactive jump instructions (00640, 00650).

5.3.1.2 Full bit

The full bit is set (channel full) whenever a word is written into the data register by either a PP or an external device. The full bit is cleared whenever a word is read from the data register by either a PP or an external device, or when the channel goes inactive.

The PP sets the full bit during the execution of the output instructions (00720, 00721, 0073X, 1073X) once for each word written. Either the external device (previously conditioned for output) or another PP performing an input clears the full bit when it recognizes the full condition and reads the word from the data register.

A PP also sets the full bit when a function instruction is executed (00760, 00770).

The state of the full bit is sensed from a PP by the use of the full/empty jump instructions (00660, 00670)

5.3.1.3 Flag bit

The flag bit is set or cleared only by PP instructions (00641, 00651). The state of the flag bit is sensed from a PP by the use of the flag set/clear jump instructions (1064X, 1065X).

The flag bit is intended to provide dual PP I/O drivers with a synchronization mechanism. As such, the flag condition cannot be altered from an external device.

5.3.1.4 Channel Error Flag

The Channel Error Flag is set:

a) When a parity error is detected on the data in the Channel Register when it is full, and

b) When the error-in line on the CYBER 180 channel is pulsed by an external device.

The Channel Error Flag is sensed by a PP through use of the channel error flag test and clear instructions (00661 and 006671).

5.3.2 Real Time Clock

A real time clock is available on channel 14B for CYBER 170 compatibility. This clock is a *12-bit register* (16-bit register on I0) that is incremented once every microsecond. No overflow capability is provided and the register may be neither set nor cleared.

5.3.3 Two Port Multiplexer

5.3.3.1 General Description

The Two Port Multiplexer (Two Port Mux) interfaces IOU Channel 15 with two asynchronous RS-232-C communications interfaces. Each port (port 0 and port 1) has an eight position Baud Rate Selector switch [110, 300, 600, 1200, 2400, 4800, 9600 and 19200 (not available on I2) baud] and a four position Port Options keylock switch. The S0 selects the port baud rate through auto speed recognition on logins and through a Two Port Mux function for PP initiated port communications. The S0 does not have an external Baud Rate Selector switch.

Each port has a 64 character output buffer and a 16 character input buffer (1 character for I2).

Special features supported by the Two Port Mux are:

- Auto Answer
- Remote Power Control
- Remote Deadstart
- Auto Dial-out
- Calendar Clock
- Save PP registers across deadstart
- Save Channel Status across deadstart

There is an RS-366A interface on port 1 only. Therefore, the Auto Dial-Out feature is supported on port 1 only. The I0 Two Port Mutiplexer does not have an RS-366A interface.

The following table lists the devices which the two port multiplexer can use to display the system deadstart panel settings:

		I0	I1	I1CR	I2	I4	Dual I4	I4C
Two	CC545 system console		X	X		X		
Port	CDC 752/722 terminal		X	X				
Mux	CDC 721 terminal (CC634B)			X		X		
display	PC console (CC595)	X						
	PC console (CC598B)					X	X	
Switch panel display					X			
Dedicated load device console (CC598A)								X

NOTE: The following capabilities are not available on the I2 Two Port Mux.

- Local/Remote Deadstart
- Remote Power Control
- Port Options keylock switch
- RS-232 Loopback
- Auto Dial-out
- Calendar Clock
- Save PP registers across deadstart
- Save Channel Status across deadstart

NOTE: The following capabilities are not available on the I1 Two Port Mux.

- Save PP registers across deadstart
- Save Channel Status across deadstart

NOTE: The following capabilities are not available on the I0 Two Port Mux:

- Auto Dial-out
- Auto Answer
- Save PP registers across deadstart
- Save Channel Status across deadstart

5.3.3.2 Interface Definitions

5.3.3.2.1 Channel 15B to PPs

The interface between channel 15B and the PPs is identical to the user channel to PP interface. All I/O instructions can be performed on channel 15B. The internal channel data path is 16 bits wide.

5.3.3.2.2 RS-232 Interface

Both ports of the Two Port Mux use the EIA Standard RS-232-C asynchronous transmission scheme.

The following signals are available on each port:

Transmitted Data
Data Terminal Ready
Request to Send

Received Data
Data Set Ready
Clear to Send
Carrier On
Ring Indicator

Signal Definitions:

Transmitted Data

This signal is used for the serial transmission of data from the Two Port Mux to external data terminal equipment.

Data Terminal Ready

This signal is used to indicate a request by the Two Port Mux to connect to the external data terminal equipment.

Request to Send

This signal is used to indicate a request by the Two Port Mux for the external data terminal equipment to prepare for data transmission from the Two Port Mux.

Received Data

This signal is used for the serial transmission of data from the external data terminal equipment to the Two Port Mux.

Data Set Ready

This signal is used to indicate to the Two Port Mux that the external data terminal equipment is connected and ready for use. This signal is a response by the external data terminal equipment to the Data Terminal Ready signal from the Two Port Mux.

Clear to Send

This signal is used to indicate to the Two Port Mux that the external data terminal equipment is ready to receive data. This signal is a response by the external data terminal equipment to the Request to Send signal from the Two Port Mux.

Carrier On

This signal is used to indicate to the Two Port Mux that the external data terminal equipment (modem) is receiving a signal which meets it suitability criteria.

Ring Indicator

This signal is used to indicate to the Two Port Mux that the external data terminal equipment (modem) is receiving a ringing signal on its communication channel. This signal is not disabled by the OFF condition of the Data Terminal Ready signal.

5.3.3.2.3 RS-366A Interface

NOTE: The RS-366A interface is not available on the I0 Two Port Mux.

The following signals are available:

Call Request
Digit Present
Four Data Bits
Power Indication
Data Line Occupied
Present Next Digit
Abandon Call
Call Origination Status

Signal Definition:

Call Request

The Two Port Mux generates this signal to request the automatic calling equipment to originate a call. This signal must be maintained in the ON condition until the Call Origination Status signal is turned on or else the call is aborted.

Digit Present

The Two Port Mux generates this signal to indicate that the automatic calling equipment may read the Data bits.

Four Data Bits

The Two Port Mux generates these four binary data signals as a code to the automatic calling equipments.

DIGIT	DATA SIGNALS			
	NB8	NB4	NB2	NB1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
*	1	0	1	0
#	1	0	1	1
End of Number	1	1	0	0
Second Dial Tone	1	1	0	1
Unassigned	1	1	1	0
Unassigned	1	1	1	1

Power Indication

The automatic calling equipment generates this signal to indicate to the Two Port Mux that power is on in the automatic calling equipment.

Data Line Occupied

The automatic calling equipment generates this signal to indicate to the Two Port Mux that the communication channel is in use.

Present Next Digit

The automatic calling equipment generates this signal to indicate to the Two Port Mux that the automatic calling equipment is ready to accept the next digit.

Abandon Call

The automatic calling equipment generates this signal to indicate to the Two Port Mux that the connection to a remote data station is probably not successful and is a suggestion to the Two Port Mux to abandon the call. This signal by itself does not abandon the call.

Call Origination Status

The automatic calling equipment generates this signal to indicate to the Two Port Mux that the automatic calling equipment has completed its call functions and that control of the communication channel has been transferred from the RS-366A interface to the RS-232-C interface.

5.3.3.3 Characteristics

5.3.3.3.1 PP to Two Port Mux Function Codes

The Two Port Mux uses the least significant 12 bits of data from channel 15B as the function code. A 12 bit function word from the PP is translated to specify the operating condition of the Two Port Mux as shown below in octal. The Two Port Mux responds with an Inactive-In signal to any function code received from channel 15B as long as one of the two ports is selected. If the function code is a "Not Used" code, the Two Port Mux remains selected but is not set up for any operation.

CODE	FUNCTION
7XX0	Select Port 0
7XX1	Select Port 1
6XXX	Deselect Two Port Mux
1X02 †	Read deadstart port/terminal type
1X03 * ‡	Set port baud rate
1X04 *	Read Calendar Clock
1X05 *	Write Calendar Clock
1X06 * ‡	Write Auto Dial-Out Data
1X07 * ‡	Read Auto Dial-Out status
1X10 * ‡	Abandon Call
1X20-26 † ‡	Read pre-deadstart copies of PP register
1X27 † ‡	Read pre-deadstart copies of Channel Status
00XX	Read Two Port Mux status
01XX	Read port data
02XX	Write port data
03YY ‡	Set Port operation mode
04X0 ‡	Clear Data Terminal Ready signal
04X1 ‡	Set Data Terminal Ready signal
05X0 ‡	Clear Request To Send signal
05X1 ‡	Set Request To Send signal
07XX	Clear Output and Input Buffers

* Not available on I2
† Not available or either I1 or I2
‡ Not available on I0

Notes:

(1) X - Don't care bits
(2) In the least significant octal number of the 7XX0, 7XX1, 04X0, 04X1, 05X0 and 05X1 function codes, only bit 63 is used. Bits 62 and 61 are don't care bits.

Function Code Definition:

Select (7XX0,7XX1)

The Select function code is used to select one of the two ports. The port number (zero or one) is specified by the least significant bit of the Select function code. Once a port is selected, all subsequent function codes and data are sent to that port only. If one of the ports was previously selected and a Select function code is issued to select the unselected port, the result is that the previously selected port is deselected and the previously unselected port is selected. The Two Port Mux always responds to the Select function code with an Inactive signal to channel 15B.

Read Deadstart Port/Terminal Type (1X02)

This function is used by a PP to identify the port which initiated the last deadstart operation. Also returned is the class of terminal which is connected at that port. The function (1X02) is sent to the Two Port Mux by the PP and in response a word is returned for the PP to input. The format of the response word is:

```
56                                  63
| T | T | T | T | P | P | P | P |

where port number PPPP = 0        Channel 10B
   (bits 60-63)          1        Port 0
                         2        Port 1
                         3-F(16)  Not assigned
and terminal type TTTT = 0        CC545
   (bits 56-59)          1        CDC 752/CDC 722 (Viking TTY)
                         2        CDC 721 (Viking X)
                         3        PC Console
                         4        Dedicated Load Device (DLD) Console
                         5-F(16)  Not assigned
```

It is the responsibility of the Two Port Mux (excluding I2) to determine and store the terminal type and port number of the logged-in deadstart device. The actual implementation can be by automatic terminal detection or by directly inquiring the terminal operator.

Set Port Baud Rate (1X03)

This function is used by a PP to set the baud rate of the port which is currently selected. The function (1X03) is sent to the Two Port Mux by the PP followed by an Active-out signal, a baud rate selection word and an Inactive-out signal.

The format of the baud rate selection word is:

56							63
0	0	0	0	R	R	R	R

```
where rate RRRR = 1    110 baud
   (bits 60-63)   2    300
                  3    600
                  4   1200
                  5   2400
                  6   4800
                  7   9600
                  8  19200
                  9  38400 (I4 & I4C only)
```

All other codes are illegal and will be ignored by the Two Port Mux.

Read Calendar Clock (1X04)

This function is used to read the Calendar Clock. A Two Port Mux Select function to either Port 0 or Port 1 is needed to read the Calendar Clock. After channel 15B has sent this function (1X04) and an Active-out signal, the Two Port Mux responds with eight Full signals to channel 15B, each accompanied by an eight bit word of data in the format shown below. This input sequence is terminated with an Inactive signal from channel 15B (if less than eight words are read, the effects on channel 15B are undefined).

WORD	56	57	58	59	60	61	62	63
				Status		Information		
0	0	0	0	0	0	0	0	S
	Tens Of Years				Units Of Years			
1	Y	Y	Y	Y	y	y	y	y
	Tens Of Months				Units Of Months			
2	0	0	0	M	m	m	m	m
	Tens Of Days				Units Of Days			
3	0	0	D	0	d	d	d	d
	Tens Of Hours				Units Of Hours			
4	0	0	H	H	h	h	h	h
	Tens Of Minutes				Units Of Minutes			
5	0	M	M	M	m	m	m	m
	Tens Of Seconds				Units Of Seconds			
6	0	S	S	S	s	s	s	s
	Reserved For Future Use							
7	0	0	0	0	0	0	0	0

Note: The status word will have bit 63 (LSB) as the bit which when set, means that the "Wall Clock Time Integrity Has Been Lost." (i.e. there has been a power failure and the clock data is no longer valid.)

Note: On I0, if the clock is malfunctioning and the Two Port Mux cannot read it, the Two Port Mux will disconnect channel 15B.

Write Calendar Clock (1X05)

This function is used to write (set) the Calendar Clock. A Two Port Mux Select function to either Port 0 or Port 1 is required to write to the Calendar Clock. This function (1X05), followed by an Active-out signal tells the Two Port Mux to treat channel 15 Data-out as data to set the calendar time (see data format below). All six words must be written each time the Calendar Clock is set. If this output sequence is terminated early with an Inactive-out signal from channel 15B (less than six words written), the effects on the Two Port Mux and the Calendar Clock are undefined. The smallest time unit that can be set is the unit of minutes. Tenth's of seconds, Units of seconds and Tens of seconds are set to zero.

WORD	Status				Information			
	Tens of Years				Units of Years			
0	Y	Y	Y	Y	y	y	y	y
	Tens of Months				Units of Months			
1	0	0	0	M	m	m	m	m
	Tens Of Days				Units Of Days			
2	0	0	D	D	d	d	d	d
	Tens of Hours				Units of Hours			
3	0	0	H	H	h	h	h	h
	Tens of Minutes				Units of Minutes			
4	0	M	M	M	m	m	m	m
	Reserved For Future Use							
5	0	0	0	0	0	0	0	0

Note: Writing the wall clock automatically resets bit 63 of the status word to a 0 value to indicate that the wall clock data is now valid.

Write Auto Dial-Out Data (1X06)

This function is used to set up the Two Port Mux for an Auto Dial-Out operation. Port 1 must be selected before this function is issued. This function (1X06) followed by an Active-out signal, sets up the Two Port Mux to treat channel 15B Data-Out as two digits to be passed on to the calling automatic equipment (See data format below). An "End Of Number" code must follow the last telephone number in the data. If there is an even number of digits in the telephone number, the End Of Number code will be in the four most significant bits of the last word and the four least significant bits are don't care bits. An Inactive-out signal from channel 15 to the Two Port Mux following the data-out operation initiates the Auto Dial-Out operation between the Two Port Mux and the automatic dailing equipment. Now the PP's should set "Data Terminal Ready" on Port 1 so that the automatic calling equipment can transfer control to the RS-232 interface when the dailing is completed. While the Two Port Mux is "dialing", the PP's may monitor the status of the call by looking at the "Abandon Call" and "Call Origination Status" status bits.

This function (1X06) should not be issued unless the Auto Dial-Out status bit "Power Indication" is a "one" and the Auto Dial-Out status bit "Data Line Occupied" is a "zero".

Bits	Description
56-59	First number
60-63	Second number

Also see data code table in RS-366A Interface section.

Read Auto Dial-Out Status (1X07)

This function is used to request the Two Port Mux for an Auto Dial-Out status operation. Port 1 must be selected before this function is issued. After channel 15B has sent this function (1X07) and an Active-out signal, the Two Port Mux responds with only one Full signal accompanied by a status word shown below.

Bits	Description
52-59	Not Used
60	Abandon Call
61	Call Origination Status
62	Data Line Occupied
63	Power Indication

For a description of these signals see the RS-366A interface section of this document.

Abandon Call (1X10)

This function is used to tell the Two Port Mux to abandon the call currently being attempted. It would normally be used when a PP has sensed the "Abandon Call" status bit during the dailing operation. The Two Port Mux will clear the "Call Request" signal to the automactic calling equipment on receipt of this function.

Read Pre-deadstart Copies of P, Q, K and A Registers (1X20-26) (I4 Only)

These functions are used to read the values of the P, Q, K and A registers that are stored in Two Port Mux RAM immediately prior to either short or long deadstart. Either Port 0 or Port 1 must be selected. The logical barrel desired is specified as shown below:

1X20 = NIO Barrel 0
1X21 = NIO Barrel 1
1X22 = NIO Barrel 2
1X23 = NIO Barrel 3
1X24 = CIO Barrel 0
1X25 = CIO Barrel 1
1X26 = All Barrels

All logical barrel and PP numbers mentioned here refer to the configuration of the system effective at the time the registers are sampled. This may or may not be the same as the configuration immediately after the short or long deadstart.

The values stored in RAM include two samples of each register, one sample taken approximately 610 milliseconds after the other. The first set is taken with the PPs running; the second set is taken after the PPs have been idled. Note that the Two Port Mux does not read the P, Q, K and A registers simultaneously. With the PPs running, this means that the captured contents of one register will not necessarily bear any relationship to the captured contents of another, unless the PP is hung. With the PPs idled the contents of P, Q and A should correlate. K will be 1077B for an idled PP.

After channel 15B has sent this function and an Active-out signal, the Two Port Mux responds by sending 90 bytes of data if only one barrel was requested, or 540 bytes if all barrels were requested. Each byte is transmitted over bits 56-63 of channel 15 (bits 48-55 are zero) and is accompanied by a Full signal. For all barrels that are not installed, the Two Port Mux still returns the proper number of bytes, but each byte is zero. The Two Port Mux terminates the transfer by sending an Inactive signal after returning the requested number of bytes. If channel 15 attempts to terminate the sequence early by sending an Inactive-out signal, no more than 4 microseconds may elapse between the time the Two Port Mux sends the last Full signal and the time it receives the Inactive-out signal or it will transmit another byte.

The Two Port Mux transmits the data to channel 15 in the order shown below:

Logical NIO Barrel 0, Logical PP 0:

P Register - MSB
P Register - LSB
Q Register - MSB
Q Register - LSB
K Register - MSB
K Register - LSB
A Register - MS 2 bits
A Register - 2nd LSB
A Register - LSB
P'Register - MSB
P'Register - LSB
Q'Register - MSB
Q'Register - LSB
K'Register - MSB
K'Register - LSB
A'Register - MS 2 bits
A'Register - 2nd LSB
A'Register - LSB

Logical NIO Barrel 0, Logical PP 1:

Logical NIO Barrel 0, Logical PP 2:

Logical NIO Barrel 0, Logical PP 2:

Logical NIO Barrel 0, Logical PP 4:

Logical NIO Barrel 1:

Logical NIO Barrel 2:

Logical NIO Barrel 3:

Logical CIO Barrel 0:

Logical CIO Barrel 1:

Read Pre-deadstart Copies of Channel Status (1X27) (I4 only)

This function is used to read the copies of NIO and CIO channel status stored in Two Port Mux RAM immediately prior to either short or long deadstart. Either Port 0 or Port 1 must be selected.

The Two Port Mux takes the channel status after all PPs have been idled and the second sample of the P, Q, K and A registers has been taken. (See functions 1X20-27.) For this reason, the copies of channel status will correlate with this second set of P, Q, K and A register samples.

After channel 15B has sent this function and an Active-out signal, the Two Port Mux responds by sending 38 bytes of data, each byte containing the status of one channel. For all channels not installed, the Two Port Mux returns a byte of all zeros. The Two Port Mux sends each byte accompanied with a Full signal over bits 56-63 of channel 15 (bits 48-55 are zero) and terminates the transfer by sending an Inactive signal. If channel 15 attempts to terminate the sequence early by sending an Inactive-out signal, no more than 4 microseconds may elapse between the time the Two Port Mux sends the last Full signal and the the time it receives the Inactive-out signal or it will transmit another byte.

The Two Port Mux transmits the data to channel 15 in the order shown below:

NIO Channel 0 Status byte
NIO Channel 1 Status byte
↓ ↓
↓ ↓
NIO Channel 33 Status byte
CIO Channel 0 Status byte
↓ ↓
↓ ↓
CIO Channel 11 Status byte

Each byte is defined as follows:

Bit 63: channel ERROR
Bit 62: channel FLAG
Bit 61: channel FULL
Bit 60: channel ACTIVE
Bit 59: Parity Error Disabled (NIO only)
Bit 58-56: reserved

Deselect Two Port Mux (6XXX)

This function is used to deselect from channel 15B any ports of the Two Port Mux that may have been selected. After this function has been issued, channel 15B may be used for PP to PP communication. Deselecting a port does not affect any operations going on between the Two Port Mux and external equipment. For example, if an output operation (02XX function) has just been performed and the 64-character output buffer is full, the Two Port Mux will continue to empty the buffer even though the port has been deselected.

Inter-PP communication over channel 15B on I0 should be used with caution since the transfer rate can vary in the range from 5 microseconds/word to 1 millisecond/word (100 microseconds/word is typical) depending upon Two Port Multiplexer microprocessor activity.

Read Two Port Mux Status (00XX)

This function is used to request the Two Port Mux for an RS-232 port status operation. After channel 15B has sent this function (00XX) and an Active-out signal, the Two Port Mux responds with only one Full signal accompanied by a status word shown below.

Bits	Description
52-57	Not Used (zero)
58	Call Origination Status
59	Output Buffer not full
60	Input ready
61	Carrier On
62	Data Set Ready
63	Ring Indication

- Call Origination Status (Bit 58)

  Bit 58 is not used for I0. Except for S0, this bit is used on Port 1 only. This bit is a zero on Port 0. See the RS-366A interface section of this document for a description of this status bit.

- Output Buffer Not Full (Bit 59)

  Except for S0, this bit is used to indicate that the 64 character output buffer for the selected port has less than 64 characters in it.

  For S0, this bit being set indicates that the output buffer can accept a minimum of 100 characters. For I0, the output buffer is a variable length. On a disconnected port, bit 59 will always indicate a full buffer.

- Input Ready (Bit 60)

  This bit is used to indicate that the selected port has data ready to input to a PP.

  See section on RS-232 interface definition for a description of the remaining status bits.

  For I0, bit 60 indicates that the selected port has character data to input to a PP. Packet data present is indicated in a new function.

- Data Set Ready (Bit 62)

  For I0, if a local console is present (bit 62 set), Carrier On (bit 61) will also be set (required by some PP drivers).

Read Port Data (01XX)

This function is used to request the Two Port Mux for a data input operation. After channel 15B has sent this function (01XX) and an Active-out signal, the Two Port Mux responds with only one Full signal accompanied by a data word shown below.

Bits	Description
52	Data Set Ready
53	Data Set Ready and Carrier On
54	Data-In Overrun
55	Data-In Framing or Parity Error
56-63	Data-In Bits

- Data Set Ready (Bit 52)

  For I0, if a local console is present (bit 52 set), Data Set Ready and Carrier On (bit 53) will also be set (required by some PP drivers).

- Data-In Overrun (Bit 54)

  This bit is used to indicate that the selected port has lost input data due to data coming in faster than the PP takes it. The data that accompanied this bit is the last data that the Two Port Mux received and it was written over the original data in the input buffer.

- Data-In Framing or Parity Error (Bit 55)

  This bit is used to indicate that the selected port has detected a parity error or a framing error on the data received from an external device.

- Data-In Bits (Bits 56-63)

  These eight bits are data received on the selected port when the "Input Ready" status bit (bit 60) is set. These bits are undefined when "Input Ready" is not set.

See section on RS-232 interface definition for a description of bits 52 and 53.

Write Port Data (02XX)

This function is used to request the Two Port Mux for a data output operation. The data output operation is initiated by channel 15B sending function code (02XX) and an Active-out signal. If the output buffer of the selected port is full, the Two Port Mux will respond with an Inactive signal to channel 15B. If the output buffer is not full, the Two Port Mux will send an Empty signal to channel 15B for each Full signal received from channel 15B until the output buffer becomes full. Each data word is eight bits (56-63).

Except for I0, the Full signal from channel 15B that caused the output buffer to become full shall cause the Two Port Mux to respond with an Inactive signal instead of an Empty signal. On I0, the Two Port Mux will disconnect the channel when the buffer is full, and the word on the channel will be discarded. If the port is not connected, the Two Port Mux will disconnect the channel.

The Two Port Mux sets Request To Send and Data Terminal Ready signals and begins transferring data from the output buffer to the RS-232 interface as soon as one character is in the output buffer and continues this transfer until the output buffer is empty even though the port may be deselected. A "07XX" function will terminate this data transfer (see "07XX" function).

Except for I0, the Request To Send and Data Terminal Ready signals are cleared by the Two Port Mux when the output buffer goes empty if these signals were not set previously by a "Set Data Terminal Ready" function (04X1) and "Set Request To Send" function (05X1).

Set Two Port Mux Operation Mode (03YY)

This function is used to specify the Two Port Mux Operation Mode according to bits 58-63 as shown below:

Bit	**Description**
58	ENABLE LOOP BACK
	This bit, when set, enables a roundtrip data path from channel 15B to the selected RS-232 port (UART chip) and back to channel 15B. When in Loop Back mode, the UART chip does not transmit data externally. See programming considerations for the protocol of this function. This function is not available on the I2.
59	DISABLE PARITY BIT
	When this bit is set, no parity bit is transmitted out of the selected RS-232 port and parity checking on the input data is disabled. The Stop bit(s) will immediately follow the last data bit.
60	SELECT NUMBER OF STOP BITS
	This bit selects the number of Stop bits. When this bit is clear one Stop bit is used. When this bit is set two Stop bits are used.
61,62	SELECT NUMBER OF BITS/CHARACTER

Bit 61	Bit 62	Bit/Character
0	0	5
0	1	6
1	0	7
1	1	8

63	SELECT ODD/EVEN PARITY MODE
	This bit selects the type of parity to be transmitted and also the type of parity expected in the input data. When this bit is set, even parity mode is selected. When this bit is clear, odd parity mode is selected.

Clear Data Terminal Ready (04X0)

This function is used to clear the Data Terminal Ready signal for the selected port. See RS-232 interface definition for a description of this signal.

Set Data Terminal Ready (04X1)

This function is used to set the Data Terminal Ready signal for the selected port. See RS-232 interface definition for a description of this signal.

Clear Request To Send (05X0)

This function is used to clear the Request To Send signal for the selected port. See RS-232 interface definition for a description of this signal.

Set Request To Send (05X1)

This function is used to set the Request To Send signal for the selected port. See RS-232 interface definition for a description of this signal.

Clear Output and Input Buffers (07XX)

This function is used to clear the output and input data buffers on the selected port.

5.3.3.3.2 External Device to Two Port Mux Functions

This feature is not implemented on I0.

The Two Port Mux monitors incoming signals and performs the functions described below. There are two mechanism's for alerting the Two Port Mux from an external device. The Ring Indicator signal is generated by dialing the telephone number for the computer Two Port Mux. The second mechanism is to send the ASCII code sequence for CTRL/G (first alert signal - ASCII code=07H) and CTRL/R (second alert signal - ASCII code=12H) to either port of the Two Port Mux. The action taken by the Two Port Mux depends on the position of the Port Option switch on each port. The Baud Rate switches and the Port Option switches are sampled by the Two Port Mux when the Deadstart button on the CC545 display console is pressed or when a Ring Indication signal is received on either Port 0 or Port 1. The Port Option switch for a port is also sampled when the code sequence for CTRL/G is sensed on the input to the port.

The Port Option switches define which external functions are enabled in the Two Port Mux.

Switch Position	Meaning
DISABLED	The port is disabled for all input and output operations. No data can be sent out and all incoming signals are ignored.
MSG ONLY	The port is enabled for system originated functions only. These functions may be output or input operations. Remote Power-control and Remote Deadstart are disabled.
DS ENABLED	The port is enabled for all functions except Remote Power-control.
DS/PWR ENABLED	The port is enabled for all functions (system originated, Remote Power-control and Remote Deadstart).

Any CRT terminals used on the Two Port Mux where the Deadstart Display is to be used, must be in page mode and have X-Y positioning enabled.

5.3.3.3.3 Auto Answer

This feature is not implemented on I0.

- The "Ring Indicator" signal is used for Auto Answer on both Port 0 and Port 1.
- Auto Answer is supported reqardless of whether the 400 Hz power to the mainframe is on or off (50/60 Hz power must be on).
- Auto Answer is disabled if the Port Option switch is in the "Disabled" position.
- Either the PP's or the Two Port Mux (microprocessor) may "answer" a call. If the Port Option switch is in the "MSG ONLY" position, only the PP's may answer. If the Port Option switch is in the "DS ENABLED" or "DS/PWR ENABLED" positions, the PP's have first chance to answer. If the PP's don't answer within about one second from the time that the Ring Indicator signal went to the ON state, the Two Port Mux will answer. The Two Port Mux will then ignore any attempt by the PP's to communicate on the Port that the Ring Indicator came in on until the Two Port Mux sends the "ILLEGAL TERMINAL" message because of a incorrect password or when a short deadstart sequence is completed after the Deadstart Display has been up.

5.3.3.3.4 Remote Power Control

This feature is not implemented on I0.

Remote Power Control allows control of the 400 Hz power for the system by commands from a terminal connected to Port 0 or Port 1 either directly or through a modem. Remote Power Control is implemented with Two Port Mux hardware connected by a cable to the computer power control box (p/n 11899142).

Security features to protect against invalid use of this feature are:

- The computer power control box must be set up to enable Remote Power Control.
- The Port Option switches on the mainframe must be set up to enable this feature. The Port Option switches are keylock switches where the key may be removed in any position.
- On Port 1 the user must be able to enter a correct power control password (up to 15 characters long).
- Each time the Ring Indicator is sensed by the Two Port Mux on Port 1, a new session is initiated and therefore password(s) must be entered by the user.
- If the password is not entered correctly in three attempts, the terminal is declared illegal and the connection to the terminal is dropped.
- On Port 0 no password is required for Remote Power-on, but a password is required for Remote Power-off. Remote Power-off always requires a password.
- The password may be entered or changed only from the CC545 console (if one is present) or the Port 0 terminal. The password is entered with a command from the Deadstart display.

PW PC XXXXXXXXXXXXXXX - Set Power Control password

where "XXXXXXXXXXXXXXX" is the password (1 - 15 characters long).

If the user is at a terminal using a modem to interface to port 1, Remote Power Control is acheived with the commands listed below.

Note that in these sequences:

OPR = Operator activities

TPM = Two Port Mux activities

<u>Commands</u>

OPR Call the computer (Ring Indicator).

TPM Sense Ring Indicator signal. If power is on, wait approximately one second for the PPs to answer (see auto answer). If the PP's don't answer, set Data Terminal Ready and Request To Send and send message "ENTER DEADSTART PASSWORD".

OPR Enter the Deadstart password.

TPM Check for correct password. If it is correct, send the Deadstart Display.

If the power is off:

TPM Send message "POWER IS OFF "DO YOU WANT POWER ON? Y/N"

The Two Port Mux will wait approximately 10 seconds for an operator response. If no response comes within the 10 seconds, the Two Port Mux will blank the screen and terminate the session.

OPR Enter "Y".

TPM Send message "ENTER POWER CONTROL PASSWORD".

Note: If the operator enters anything other than "Y", the Two Port Mux will blank the screen and terminate the session.

OPR Enter the Power Control password.

TPM Send message "POWER-UP INITIATED". Send the Deadstart Display when Power-up is complete.

Note: If the power-up is not completed within approximately 30 seconds, the Two Port Mux will send the message "POWER IS NOT ON - BYE" and terminate the session.

If a PP answers the call, it means the power is on. To turn the power off, a command from the Deadstart Display must be used as shown below.

OPR Press "CTRL" and "G".

TPM Sense "CTRL" and "G" code.

This code is the first alert signal to the Two Port Mux that an external command may be desired. The Two Port Mux will discontinue support of PP originated functions on both ports, set the first alert flag and send message "EXTERNAL FUNCTIONS ENABLED". After the message has been sent the Two Port Mux will wait approximately ten seconds for the second alert signal. If the second alert signal is not sensed in ten seconds, the Two Port Mux will reset the first alert flag and continue support of PP originated functions to either port.

OPR Press "CTRL" and "R".

TPM Sense "CTRL" and "R" code and take control of the port. PP originated operations on this port are discontinued. An "ENTER DEADSTART PASSWORD" message is sent to the terminal.

OPR Enter Deadstart password.

TPM Check for correct password. If the correct password is entered, send the Deadstart Display.

OPR Enter "OFF PWR".

TPM Send message "ENTER POWER CONTROL PASSWORD".

OPR Enter the Power Control password.

TPM Check for correct password. If it is correct, open the power control relay and send message "POWER-DOWN INITIATED". When the power has dropped, send message "POWER IS OFF."

The procedure for a terminal on Port 0 is the same as for Port 1 except no passwords are required for powering-up or getting the Deadstart display. A password is still required for powering-down.

Note: If the user enters an invalid password, the Two Port Mux will send the message "INVALID PASSWORD / TRY AGAIN". If the password is still wrong on the third try, the Two Port Mux will send the message "ILLEGAL TERMINAL" and clear Request To Send and Data Terminal Ready signals.

After the password has been entered correctly, the Two Port Mux will not ask for a password until the Two Port Mux senses the next Ring Indicator signal.

Note: If the operator tries to use a feature which has not been enabled via the Port Option Switch, the Two Port Mux will send the message "SWITCH IN DISABLED POSITION" and continue support of PP originated functions to either port.

5.3.3.3.5 Remote Deadstart

This feature is not implemented on I0.

The Remote Deadstart feature provides the cabability to deadstart the PP's from a terminal on either Port 0 or Port 1 of the Two Port Mux. A Deadstart password is required to use this feature on Port 1. No security is provided for Port 0.

The Deadstart password has the same security features as the power control password and is set from the Deadstart display with the following command:

PW DS XXXXXXXXXXXXXXX - Set Deadstart Password

where "XXXXXXXXXXXXXXX" is the password (1 - 15 characters long).

The protocal for using this feature is the same as for the Remote Power-control. First get the Deadstart display using the methods shown under Remote Power Control. Once the Deadstart display is up, the user may Deadstart the IOU by entering "L" to initiate a Long Deadstart sequence or "S" to initiate a Short Deadstart sequence. All the maintenance features associated with the Deadstart display are also available.

Note: When the Deadstart Display is active on either port, the Two Port Mux does not support PP originated functions to either port.

EXAMPLES (Assume that the PP's do not answer the call and the required feature is enabled via the option switch):

POWER IS ON	
PORT 0	PORT 1
OPR Call computer TPM Send Oeadstart display	OPR Call computer TPM "ENTER OEAOSTART PASSWORO" OPR Enter password TPM Send Oeadstart display

Both Port 0 and Port 1
OPR "L" (Long Deadstart) OPR "CTRL/G" TPM "EXTERNAL FUNCTIONS ENABLEO" OPR "CTRL/R" TPM Send Oeadstart display OPR "OFF PWR" TPM "ENTER POWER CONTROL PASSWORO" OPR Enter password TPM "POWER-OOWN INITIATED" "POWER IS OFF"

POWER IS OFF	
PORT 0	PORT 1
OPR Call computer TPM "POWER IS OFF" "OO YOU WANT POWER ON? Y/N" OPR "Y" TPM "POWER-UP INITIATED" Send Deadstart display	OPR Call computer TPM "POWER IS OFF" "OO YOU WANT POWER ON? Y/N" OPR "Y" TPM "ENTER POWER CONTROL PASSWORD" OPR Enter password TPM "POWER-UP INITIATEO" Send Deadstart display

5.3.3.4 Performance

5.3.3.4.1 Function Response Times

The function response time is the amount of time the Two Port Mux takes to send an Inactive-in signal to channel 15B in response to a Function-out received from channel 15. Some of the functions are translated by hardware and some by microprocessor firmware. The functions translated by hardware are relatively fast, less than one major cycle (500 nanoseconds). The functions translated by firmware are relatively slow and don't have a fixed response time. This is because the Function-out Signal from channel 15B is sensed in a polling loop and because of possible interrupts occurring during the function sequence. The firmware response times range from about 50 microseconds up to 1 millisecond with a typical time of about 300 microseconds.

All I0 functions are slow functions.

All I2 functions are fast functions.

I1/I4 Fast Functions	I1/I4 Slow Functions
7XX0	1X02
7XX1	1X03
6XXX	1X04
00XX	1X05
01XX	1X06
02XX	1X07
Not Used Codes	1X10
	1X20-26*
	1X27*
	03YY
	04X0
	04X1
	05X0
	05X1
	07XX

*I4 only

5.3.3.4.2 Data Transfer Rates

Data transfer rates can be classified into two categories:

Hardware transfer rate - a word can be transfered every 1 microsecond.

Firmware transfer rate - a word can be transfered every 50 microseconds to 1 millisecond depending upon the activity of the Two Port Multiplexer microprocessor. The typical time is 300 microseconds/word.

PP to Two Port Mux

The following functions involve data transfers from the PP's to the Two Port Mux:

02XX — The "Write Port Data" function involves a data transfer between the PP's and the 64 character output buffer. A block data transfer may be used; the hardware transfer rate applies to I1 and I2 while the firmware transfer rate applies to I0.

1X05

1X06 — The "Write Calendar Clock" and "Write Auto Dial-out Data" functions involve data transfers between the PP's and the microprocessor memory. A block data transfer may be used. The firmware transfer rate applies to these functions.

Two Port Mux to PP

The following functions involve data transfers from the Two Port Mux to the PP's:

00XX — The "Read Two Port Mux Status" function is a one word input operation.

01XX — On I2 the "Read Port Data" function is a one word input operation from a hardware register. The data is available in less than 500 nanoseconds after channel 15B is made active.

On I1 the input buffer is three words in length. One of these three words is in the hardware register. The rest of the words are in a "soft" buffer in the microprocessor memory. It takes from 50 microseconds up to 1 millisecond (the typical time is 300 microseconds) for the microprocessor to transfer the next word into the hardware register and set the "Input Ready" status bit.

On I0 the entire buffer is in the microprocessor memory and the firmware transfer rate applies.

1X04	Read Calendar Clock	All of these functions involve a data transfer from the microprocessor memory to the PP's. The firmware transfer rate applies to these functions.
1X07	Read Auto Dial-out Status	
1X20-26*	Read pre-deadstart copies of PP registers	
1X27*	Read pre-deadstart copies of Channel Status	

*I4 only

RS-232 Interfaces

Baud rates of 110*, 300, 600, 1200, 2400, 4800, 9600 and 19200† are supported on both Port 0 and Port 1.

---

* unavailable on I0
† unavailable on I2

RS-366A Interface

The rate that the Two Port Mux sends dailing numbers to the automatic calling equipment depends on the automatic calling equipment.

5.3.3.4.3 Calendar Clock Accuracy

The calendar clock shall have a long term accuracy as shown below:

Model	Accuracy
I1	+1 min., −10 min. per year on 50/60 hertz power and battery backup.
All others	±1 min. per year on 50/60 hertz power; ±12 min. per year on battery backup.

5.3.3.5 Programming Considerations

5.3.3.5.1 RS-232 Interfaces

LOCAL

These programs illustrate how a terminal could be programed when it is connected directly to a Two Port Mux port with the special cable (p/n 19266318).

This program uses the block output instruction to send data to the Two Port Mux.

```
FNC   7000B,15B     Select Port 0
FNC   0304B,15B     Set 7 bits/character, odd parity and one stop bit
FNC   0000B,15B     Two Port Mux Status
ACN   15B
IAN   15B           Input status
DCN   15B
LPN   20B           Check for Output Buffer Not Full
ZJN   EXIT          If full
FNC   0200B,15B     Write port data
ACN   15B
LDD   WORDCOUNT
OAM   BUFFER,15B    Output the data
STD   SAVE          Save remaining word count
NJN   EXIT          Exit if Two Port Mux deactivated the channel
DCN   15B
UJN   EXIT
```

This program uses only the single word output instruction to send data to the Two Port Mux.

```
      FNC  7000B,15B       Select Port 0
      FNC  0304B,15B       Set 7 bits/character, odd parity and one stop bit
      FNC  0200B,15B       Write Port Data
      ACN  15B
TAG1  IJM  EXIT,15B        Exit if output buffer full
      LDM  BUFFER,INDEX    Get data from data buffer
      OAN  15B             Output the data word
      AOD  INDEX
      LMD  END             Check for end of data buffer
      NJN  TAG1            If not end of data buffer
      DCN  15B
      UJN  EXIT
```

This program reads one word from the Two Port Mux input buffer.

```
      FNC  7000B,15B       Select Port 0
      FNC  0000B,15B       Read Status
      ACN  15B
      IAN  15B             Input Status
      DCN  15B
      LPN  10B             Check for Input Ready
      ZJN  EXIT            If no Input Ready
      FNC  0100B,15B       Read port data
      ACN  15B
      IAN  15B             Input the data word
      DCN  15B
      STD  DATA
      LPC  1400B           Check for Data-in Overrun and Data-in Framing or parity error
      NJN  ERROR           If Data-in error
      UJN  EXIT
```

REMOTE

This program illustrates how a device connected to a Two Port Mux port through a modem could be programmed for auto answer.

A PP should poll the ports that have a modem connected to them for the "Ring Indicator" signal. Once Ring Indicator" is detected the PP must answer within approximately one second by setting "Data Terminal Ready". If the PP does not answer within the time limit, the microprocessor will answer the call looking for remote commands.

This program answers the phone and sets up the modem for communication to a remote device.

```
        FNC   7001B,15B     Select Port 1
        FNC   0000B,15B     Two Port Mux Status
        ACN   15B
        IAN   15B           Input the RS-232 status
        DCN   15B
        LPN   1             Check for Ring Indication
        ZJN   EXIT          If no Ring Indication
        FNC   0401B,15B     Set Data Terminal Ready
TAG1    FNC   0000b,15B     Two Port Mux Status
        ACN   15B
        IAN   15B           Input the RS-232 status
        DCN   15B
        LPN   2             Check for Data Set Ready
        ZJN   TAG1          If no Data Set Ready
        FNC   0501B,15B     Set Request To Send
TAG2    FNC   0000B,15B     Two Port Mux Status
        ACN   15B
        IAN   15B           Input the RS-232 status
        DCN   15B
        LPN   4             Check for Carrier On
        ZJN   TAG2          If no Carrier On
        UJN   EXIT
```

Output and Input

The output and input programming is similar to that used for a local terminal except that it may be desirable to monitor the "Carrier On" status while communicating over telephone lines. Also "Request To Send" and "Data Terminal Ready" should be cleared at the end of the session.

5.3.3.5.2 RS-366A Interface (Auto Dial-Out)

This program illustrates how an "Auto Dial-out" sequence could be done.

```
        FNC   7001B,15B     Select Port 1
        FNC   1007B,15B     Auto Dial-out Status
        ACN   15B
        IAN   15B           Input RS-366A status
        DCN   15B
        LPN   3             Check for "Power Indication and "Data Line Occupied"
        LMN   1
        NJN   ERROR         IF no power or line is occupied
        FNC   1006B,15B     Write Auto Dial-out Data
        ACN   15B
        LDN   4             Number of 8-bit PP words (two 4-bit telephone nos. per PP
                            word)
        OAM   BUFFER,15B    Copy numbers from PP memory to microprocessor memory
TAG1    FJM   TAG1,15B      Wait for Two Port Mux to take the last word
        DCN   15B           Starts dailing operation between the Two Port Mux and the
                            automatic calling equipment
        FNC   0401B,15B     Set Data Terminal Ready
TAG2    FNC   1007B,15B     Auto Dial-out Status
        ACN   15B
        IAN   15B           Input RS-366A status
        DCN   15B
        LPN   14B           Check for "Call Origination Status" or "Abandon Call"
        ZJN   TAG2          If call not complete or no abandon call
        LPN   10B           Check for "Abandon Call"
        NJN   TAG5          If automatic dialing equipment is requesting an abandon call
                            function
TAG3    FNC   0000B,15B     Two Port Mux Status
        ACN   15B
        IAN   15B           Input RS-232 status
        DCN   15B
        LPN   2
        ZJN   TAG3          If no "Data Set Ready"
        FNC   0501B,15B     Set "Request To Send"
TAG4    FNC   0000B,15B     Two Port Mux Status
        ACN   15B
        IAN   15B           Input RS-232 status
        DCN   15B
        LPN   4
        ZJN   TAG4          If no "Carrier On"
        UJN   EXIT
TAG5    FNC   1010B,15B     Abandon Call
        FNC   0400B,15B     Clear "Data Terminal Ready"
        UJN   RETRY
Buffer  DATA  0044B         "2","4"                   } Octal code for
        DATA  0223B         "9","3"                   } telephone number
        DATA  0003B         "0","3"                   } 249-3034.
        DATA  0114B         "4","End of Number"       }
```

5.3.3.5.3 Calendar Clock

These programs show how to set and read the Calendar Clock. Either Port 0 or Port 1 may be used when using the Calendar Clock.

Output

```
        FNC   7000B,15B     Select Port 0
        FNC   1005B,15B     Write Calendar Clock
        ACN   15B
        LDN   4
        OAM   TIME,15B      Output time to calendar clock
TAG1    FJM   TAG1,15B      Wait until Two Port Mux takes last word
        DCN   15B
        UJN   EXIT
```

Input

```
        FNC   7000B,15B     Select Port 0
        FNC   1004B,15B     Read Calendar Clock
        ACN   15B
        LDN   4
        IAM   TIME,15B      Read calendar clock
        DCN   15B
        UJN   EXIT
```

5.3.3.5.4 Loop Back

This maintenance feature can be used to check the round trip data path from the PP's out to either the Port 0 or Port 1 UART chip and back again to the PP's by using the program below.

```
      FNC   7000B,15B      Select Port 0
      FNC   0340B,15B      Enable Loop Back
      FNC   0200B,15B      Write Port Data
      ACN   15B
      LDN   3              Enough words to fill the input buffer
      OAM   DATA,15B       Send data to Two Port Mux
      DCN   15B
      STD   INDEX
TAG1  FNC   0000B,15B      Two Port Mux Status
      ACN   15B
      IAN   15B            Input RS-232 status
      DCN   15B
      LPN   10B            Check for "Input Ready"
      ZJN   TAG1           If no Input Ready
      FNC   0100B,15B      Read Port Data
      ACN   15B
      IAN   15B            Input data word
      DCN   15B
      LPC   377B           Mask off status bits
      LMM   DATA,INDEX     Check data
      NJN   ERROR          If data error
      AOD   INDEX
      LMN   3
      NJN   TAG1           If not done checking all three data words
      UJN   EXIT
```

5.3.4 Maintenance Channel

The C180 architecture does not require a dedicated maintenance channel. Some early C180 systems, however, contained a dedicated maintenance channel on Channel 17B. It is connected to all Maintenance Access Controls in the system, and enables any PP to perform on-line maintenance functions and to sense the status of the system. Details of this channel, its interface and control signals are documented in the model-dependent engineering specifications for the systems having this feature.

5.3.5 External Interface

5.3.5.1 General

The external interface consists of logic modules containing the desired channel protocol and electrical interface mechanisms to emulate the appropriate channel for an external device.

5.3.5.1.1 CYBER 170 External Interface

The CYBER 170 external interface uses bidirectional, synchronous communication to allow the IOU to communicate with CYBER 170 external devices. The transmission is accomplished via separate input and an output cable. Each cable transmits 13 data signals (12 data bits plus parity). Eight control signals are transmitted from a PP to an external device, and four from the external device to a PP. The signals are transmitted over coaxial cables using an AC transmission scheme (see 5.3.5.6).

External devices connected to the data and control lines may relay all signals to the next in line device. The relay is synchronized to a 10 MHz clock that is one of the control signals. This causes all devices to be synchronous with the IOU but displaced from each other by one or more 100 nanosecond clock periods. Since the signals are retransmitted at each device interface (passed on), powering off one device will disable data transmission to all devices beyond. The maximum cable length between devices is 70 feet.

A 12-bit external interface transmits data between bits 52-63 of the interface channel and the external device. On output, bits 48-51 of the internal channel word are not transmitted; on input, bits 48-51 of the internal channel word are cleared.

5.3.5.1.2 CYBER 180 External Interface

The CYBER 180 external interface uses bidirectional, asynchronous communication to allow the IOU to communicate with one external device. The transmission is accomplished via a single cable. The cable transmits 17 bidirectional data signals (16 data bits plus parity) and ten unidirectional control signals. The signals are transmitted over twisted pair lines in a differential mode. The data signals utilize a transmitter/receiver pair at each end of the line. The control signals have a single transmitter or receiver at each end of the line. The receiver terminates the line in its characteristic impedance to insure that the transmitted electrical wave front is not reflected back to the transmitter. As a result, data may be transmitted at high frequency without regard to the length of the cable.

This channel has been designed to drive a single high transfer rate controller. Multiple controllers on a single channel are not supported. The maximum channel cable length is 200 feet.

5.3.5.2 CYBER 170 *and CYBER 180* Channel Control Signals

5.3.5.2.1 Active

The active pulse is the signal that indicates the beginning of a data transmission. It is normally sent by a PP to an external device. It can be sent by an external device to a PP only on a CYBER-170 channel.

5.3.5.2.2 Inactive

The inactive pulse is sent from either a data sending or a data receiving device. The inactive pulse signifies the end of a data transmission and clears the active and full flags.

5.3.5.2.3 Full

The full pulse is sent from a data sending device to a data receiving device. The full pulse is sent at the same time that the data is transmitted. This indicates to the receiving device that it is to sample the data signals.

5.3.5.2.4 Empty

The empty pulse is sent by a data receiving device to a data sending device. The empty pulse is sent to acknowledge the receipt of a full pulse and the associated data. It indicates to the sending device that new data may be transmitted.

5.3.5.2.5 Function

The function pulse is only sent to an external device. It is used to indicate that the associated data signals are to be considered as control signals rather than data.

5.3.5.2.6 Master Clear

The master clear pulse is sent by the IOU to all external devices on the I/O channel. It indicates to those devices that all activity is to cease and initial conditions are to be restored.

5.3.5.2.7 Error (CYBER 180 Channel Only)

The error pulse is sent by an external device to the IOU. It indicates that an error was encountered by the external device.

5.3.5.2.8 10 MHz Clock (CYBER 170 Channel Only)

The 10 MHz clock is sent by the IOU to an external device. The signal consists of a pulse sent every 100 nanoseconds and is used to synchronize all external devices to the IOU.

5.3.5.2.9 1 MHz Clock (CYBER 170 Channel Only)

The 1 MHz clock is sent by the IOU to an external device. The signal consists of a pulse sent every 1 microsecond.

5.3.5.3 (Section intentionally left blank)

5.3.5.4 Data signals

The data signals are transmitted from a data sending device to a data receiving device along with the associated full pulses. Data signals in the form of control signals are also transmitted to an external device when a function pulse is sent.

5.3.5.5 PP and Channel Interaction

5.3.5.5.1 Active Bit

When the active bit is set by a PP (with either an 00740 or an 00741 instruction), an active pulse is sent. When the active bit is cleared by a PP, an inactive pulse is sent.

When an active pulse is sent by an external device, the active bit is set. When an inactive pulse is sent by an external device, the active bit is cleared.

5.3.5.5.2 Full Bit

When the full bit is set by a PP (with an 00720, 00721, 0073X or a 1073X instruction), a full pulse and data pulses for the data are sent. When the full bit is cleared by a PP (by an 00700, 00701, 0071X or a 1071X instruction), an empty pulse is sent.

When a full pulse is sent by an external device, the associate pulses set the channel data register and the full bit is set. When an empty pulse is sent by an external device, the full bit is cleared.

5.3.5.5.3 Function Instructions

When a function instruction (00760, 00761, 00770, 00771) is executed, the active and full bits are set and a word is written into the data register from the PP. This word is then transmitted from the data register to an external device. A function pulse transmitted to the external device indicates that the word is a control signal rather than data. The external device sends an inactive pulse to acknowledge the receipt of the function. This inactive pulse causes the inactive bit and the full bit to be set to zero.

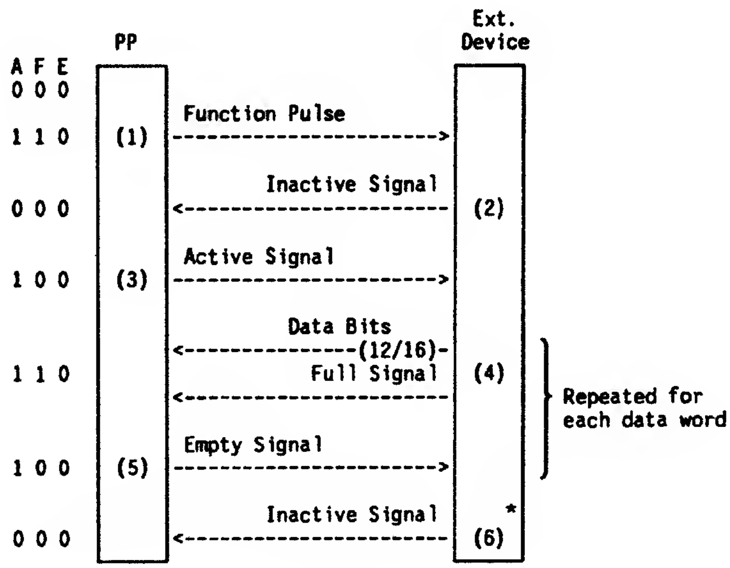
```
                                           Ext.
                PP                         Device
A F E
0 0 0      +--------+                    +--------+
           |        | Function Pulse     |        |
1 1 0      |  (1)   | ------------------------>   |
           |        |                    |        |
           |        |    Inactive Signal |        |
0 0 0      |        | <-----------------------  (2)
           |        |                    |        |
           |        | Active Signal      |        |
1 0 0      |  (3)   | ------------------------>   |
           |        |                    |        |
           |        |      Data Bits     |        |  }
           |        | -(12/16)---------------->   |  }
1 1 0      |  (4)   | Full Signal        |        |  }
           |        | ------------------------>   |  } Repeated for
           |        |                    |        |  } each data word
           |        |       Empty Signal |        |  }
1 0 0      |        | <-----------------------  (5)  }
           |        |                    |        |
           |        | Inactive Signal    |        |
0 0 0      |  (6)   | ------------------------>   |
           +--------+                    +--------+
```

<u>Data Output Sequence</u>

Key

A F E : Active, Full and Error bits

(1) PP executes a function instruction which sets the active and full bits in the internal interface, places a word in the channel register and sends a function pulse.
(2) The external device acknowledges the acceptance of the function by sending an inactive signal. This, in turn, drops the active flag and the full flag and clears the channel register.
(3) PP sets active flag to indicate that data flow is about to start.
(4) PP places either a 12-bit *or 16-bit* data word (plus parity) in the channel register, which sets the full flag and sends a full signal.
(5) The external device accepts the data word and sends an empty signal which clears the channel register and full flag.
(6) After steps (4) and (5) have been repeated a sufficient number of times to complete the data transfer, the PP drops the channel active flag which turns off the external device with an inactive signal.



Data Input Sequence

* The inactive signal is normally sent from the external device to the IOU. However, in certain cases the IOU will deactivate the channel. This is determined by the external device and the function being executed.

Key

A F E : Active, Full and Error bits

(1) PP executes a function instruction which sets the active and full bits in the internal interface, places a word in the channel register and sends a function pulse.
(2) The external device acknowledges the acceptance of the function by sending an inactive signal. This, in turn, drops the active flag and the full flag and clears the channel register.
(3) PP sets active flag to indicate that data flow is about to start.
(4) The external device reads a 12-bit *or 16-bit* word (plus parity) and sends it to the channel register with a full signal which, in turn, sets the full flag.
(5) PP stores the data word and drops the full flag which, in turn, sends an empty signal to the external device.
(6) After steps (4) and (5) have been repeated a sufficient number of times to complete the data transfer, the external device clears its active condition and sends an inactive signal to the PP. This clears the channel active flag.

5.3.5.6 Transmission Characteristics

5.3.5.6.1 CYBER 170 Channel

The transmission characteristics of the CYBER 170 channel are defined in Engineering Specification No. 19063800, "A.C. Transmission Circuit Specification for I/O Channels in the CYBER 170 System."

5.3.5.6.2 CYBER 180 Channel

5.3.5.6.2.1 Signal

Each differential signal is transmitted with voltage levels of 0.0 V (logical 0) and -0.8 V (logical 1).

5.3.5.6.2.2 Cable

The CDC 3000 Series cable (CDC part number 10382829) is used for the CYBER 180 External Interface. This cable consists of 29 twisted pair signal conducters , and a shield. The cable has a 61-pin male connector on each end. The cable propagation delay is 5.25 nanoseconds/metre (1.6 nanoseconds/foot). The characteristic impedance of the twisted pair line is 102 ohms. The signals are summarized in Table 5.3-1.

Signal Name	Connector Pins
Data Bit 2**0 (bidirectional)	A1/A2
Data Bit 2**1 (bidirectional)	A3/A4
Data Bit 2**2 (bidirectional)	A5/A6
Data Bit 2**3 (bidirectional)	A7/A8
Data Bit 2**4 (bidirectional)	A9/A10
Data Bit 2**5 (bidirectional)	B1/B2
Data Bit 2**6 (bidirectional)	B3/B4
Data Bit 2**7 (bidirectional)	B5/B6
Data Bit 2**8 (bidirectional)	B7/B8
Data Bit 2**9 (bidirectional)	B9/B10
Data Bit 2**10 (bidirectional)	C1/C2
Data Bit 2**11 (bidirectional)	C3/C4
Data Bit 2**12 (bidirectional)	C5/C6
Data Bit 2**13 (bidirectional)	C7/C8
Data Bit 2**14 (bidirectional)	C9/C10
Data Bit 2**15 (bidirectional)	D1/D2
Data Parity (bidirectional)	D3/D4
Active Out	D5/D6
Inactive Out	D7/D8
Full Out	D9/D10
Empty Out	E1/E2
Inactive In	E3/E4
Full In	E5/E6
Empty In	E7/E8
Function	E9/E10
Master Clear	F1/F2
Error In	F3/F4
Not Used	F5-F8

Table 5.3-1. CYBER 180 Channel Signal Definitions

5.3.6 Data Transmission Errors

5.3.6.1 Data-In Transmission

Data-In transmissions are checked twice for parity errors. The first check occurs when the channel register is full. Errors detected here cause the Channel Error Flag to set and, if the Fault Status Mask bit for that channel is clear, then the appropriate Channel Error bit is set in the Fault Status Register, and the Uncorrected Error and Summary Status bits are set in the Status Summary Register. Parity checking on each CYBER 170 channel may be disabled by means of a switch. The second check occurs when the PP receives the data. A parity error detection here causes the Channel Error Flag to set and causes a flag bit to set in the Fault Status Register. This is determined on a model dependent basis and details are to be found in the appropriate Engineering Specifications.

For data-in transmissions the data is always stored with regenerated, correct parity in PP memory.

5.3.6.2 Data-Out Transmissions

Data-Out transmissions are checked for parity at the channel register and, on a device dependent basis, at the receiving external device. When a parity error is detected at the channel register the Channel Error Flag is set and, if the Fault Status Mask bit for that channel is clear, then the appropriate Channel Error bit is set in the Fault Status Register, and the Uncorrected Error and Summary Status bits are set in the Status Summary Register. Parity checking on each CYBER 170 channel may be disabled by means of a switch. If the external device on the CYBER 180 channel detects a parity error, then the Error-In line will be set, which will cause the Channel Error Flag to set.

The combination of the error bits in the Fault Status Register and the Channel Error Flag permit the device drivers to detect errors and initiate recovery algorithms. In addition, these flags and error bits permit differentiation between errors arising on the channel itself, and errors arising on transmissions between a PP and the Channel register.

5.4 CACHE INVALIDATION

Note: *The following is pertinent only for those processors having cache memory.*

Cache invalidation requests are sent by the IOU to the CPU designated to be a CYBER 170 state processor. These requests are used by the CPU to perform cache purges for data words stored by the IOU. Requests are sent upon completion of the write operations in central memory under the following conditions.

5.4.1 Central Write from d to (A)

On the I2, a cache invalidation request is sent each time the 0062 instruction is executed. On the I4, a cache invalidation request is sent each time the 0062 or 1062 instructions are executed if the OS Bounds bit is set.

5.4.2 Central Write (d) Words from m to (A)

On the I2, a cache invalidation request is sent each time the address modulo 4 is equal to three and when the last word of the transfer is written.

Note: *Execution of instructions 1000, 1001, 1062, and 1063 does not invalidate the cache on I2.*

On the I4, the cache invalidation request is only sent when the OS Bounds bit is set. For instructions 0063 and 1063 the cache invalidation request is sent each time the address module 4 is equal to three and when the last word of the transfer is written.

5.5 INITIALIZATION

The system initialization process begins in the Maintenance Processor and is model dependent. It shall be activated by operator command, deadstart switch or power-on. System initialization places a minimum set of hardware in a known operational state, ready to:

a) begin Maintenance Diagnostic load.

b) begin Operating system load. The system state shall be defined by the Common Test and Initialization (CTI) Interface Specification, Doc. No. ARH2948.

c) begin utilities--dump and install.

Full initialization capability shall be available to the RTA access. The appropriate model-dependent engineering specification shall contain the detailed documentation of the initialization process.

5.6 MAINTENANCE REGISTERS

The maintenance registers are model-dependent implementations of the system maintenance strategy. The Dedicated Fault Tolerant Interface requirements are specified in section 9.0. The requirements for maintenance access are specified in section 6.0. The following functions shall be provided by the IOU. Reference the applicable engineering specification for register definition.

Register Name	Maintenance Access
Status Summary	Read
Element ID	Read
Options Installed	Read
Fault Status Mask	Read/Write
OS Bounds (1)	Read/Write
Environment Control	Read/Write
Status Register	Read
Fault Status	Read/Write
Test Mode	Read/Write

(1) Not required on C180 state only systems

Table 5.6-1. Maintenance Registers

5.6.1 OS Bounds (OSB) *(Not on I0)*

The OS Bounds Register (not implemented in I0) physically divides central memory address space into an upper and lower region for system protection during dual-state operation. An errant PP in one state then cannot alter contents of memory in the other state, thus giving some added protection to the system. A bit in the OS Bounds Register for each PP indicates into which region central memory writes and exchanges may occur. A set bit indicates the lower region:

PP CM address < OS Boundry, while a cleared bit indicates the upper region:

OS Boundry ≤ PP CM address. If the PP attempts to access its prohibited region and if the Enable OS Bounds Checking bit is set then:

- *the write or exchange will not occur*
- *the OS Bounds Fault will be set in the Fault Status Register*
- *the PP will be idled if the Enable Error Stop is set in the Environment Control Register.*

The 8-byte OS Bounds Register is updated through a maintenance channel write a byte at time. Thus one must plan for a possible indeterminent state to occur. The Enable OS Bounds Checking bit is cleared during deadstart. This will initially inhibit OS Bounds checking and allow PP's total access to central memory. The OS Bounds Register is not initialized during deadstart but must be set by software before use.

Bit Position	Byte	I2	I1
0-2	0	Not Used	Not Used
3-7	0	PPs 4-0 Barrel 0	PPs 4-0
8-10	1	Not Used	Not Used
11-15	1	PPs 4-0 Barrel 1	PPs 9-5
16-18	2	Not Used	Not Used
19-23	2	PPs 4-0 Barrel 2	Not Used
24-26	3	Not Used	Not Used
27-31	3	PPs 4-0 Barrel 3	Not Used
32-45	4,5	Not Used	Not Used
46-63	5,6,7	OS Boundry x 2**10	Same

Table 5.6-2. OS Bounds Register (MR 21)

The PPs represented in the OS Bounds Register are numbered physically, not logically. Software may use the hardware reconfiguration switches to translate from logical to physical numbering.

On the I4, the OS Bounds Register also controls cache invalidation of PP writes to central memory. (See Section 5.4)

5.7 RAM FEATURES

5.7.1 Error Detection

5.7.1.1 PP

The PP memory generates and checks parity (1 bit/word) for all data transferred between the PP memory and the PP. A parity error causes the appropriate PP memory parity error bit to be set in the IOU maintenance registers.

5.7.1.2 I/O Channels

Each parallel data channel generates and checks parity (1 bit/word) for all data transferred on the channel. A parity error causes the channel error flag and a channel parity error bit in the Fault Status Register to be set.

5.7.1.3 Central Memory Access

The central memory access generates and checks parity (1 bit/word) for each word transferred to or from central memory. A parity error causes the appropriate PP central memory error bit to be set in the IOU maintenance registers.

5.7.2 Error Recovery

The IOU maintenance registers allow a PP to be halted when a particular error condition is detected. Each error condition bit has a matching control bit. When both the error and control bits are set, the PP operation halts. The action of halting any PP due to an error condition will not halt or otherwise interfere with the operation of any other PP, unless another PP is awaiting some action to be performed by the halted PP. A software deadstart must be executed on the halted PP to regain control of that PP.

5.8 INTERFACES TO OTHER SYSTEM ELEMENTS

5.8.1 Memory/IOU

The IOU provides access to a single memory element. This interface is via a standard 64-bit memory port and utilizes a subset of the signals and functions available at the port. The IOU performs resynchronization of memory signals to the IOU clock.

5.8.1.1 Signals

Signals not described below are the same as those described in section 4.1.

5.8.1.1.1 Mark Lines

No partial write operations are performed by the IOU. These lines, therefore, will be zeroes with correct parity on read operations, and ones with correct parity on write operations.

5.8.1.2 Functions

The following memory functions are utilized by the IOU. See 4.2.

0000	*Read*
0010	*Write*
0100	*Read and Set Lock*
0101	*Read and Clear Lock*
1100	*Interrupt*

5.8.2 CPU/IOU

In the S2 system the IOU provides an interface to the CPU in the system which is designated as the CYBER 170 state CPU. This interface is used to provide cache invalidation requests and CYBER 170 exchange requests. For the S3 system, these requests are made through the memory port.

The transmission scheme shall be the standard ECL 10K DC differential scheme. The maximum wire length between transmitter and receiver shall be 15 feet.

5.8.2.1 S2 Signals

Signals from IOU to P2

Address	*21 lines + 3 lines (parity)*
Buss	*1 line*
Exchange Code	*2 lines*

Signals from P2 to IOU

Exchange accept	*1 line*
Busy	*1 line*

5.8.2.1.1 Address

The address lines transmit the CYBER 170 exchange address and the addresses for cache invalidation.

5.8.2.1.2 Buss

The buss line is used to signify that an address is being transmitted on the address lines.

5.8.2.1.3 Exchange Code

The exchange code lines carry a 2-bit code to the CPU to indicate the type of exchange:

00 = EXN (260 instruction)
01 = MXN (261 instruction)
10 = MAN (262 instruction)

An exchange code of 11 is used to indicate that the address on the address lines is a cache invalidation address.

5.8.2.1.4 Exchange Accept

The exchange accept line is used to signify that the previous exchange jump request has been honored.

5.8.2.1.5 Busy

The busy line is used to signify that the CPU cannot accept further cache addresses.

5.8.2.2 S3 Signals

Signals from M3 to P3

Address	*21 lines + 3 lines parity*
Exchange Code	*2 lines*
Invalidate	*1 line*

Signals from P3 to the IOU

Exchange Accept	*1 line*

5.8.2.2.1 Address

The address lines transmit the CYBER 170 exchange address and the addresses for each cache invalidation.

5.8.2.2.2 Exchange Code

On an exchange request the IOU sends the exchange code through the central memory port. The most significant bit of the tag field indicates an exchange request when the memory function is equal to "0000" (READ). The next two bits of the tag field indicate the type of exchange:

00 = EXN (260 instruction)
01 = MXN (261 instruction)
10 = MAN (262 instruction)

M3 sends a response back to the IOU in exactly the same manner as it would for any other read or write. These exchange codes are then routed by M3 to the P3 processor.

5.8.2.2.3 Invalidate

The invalidate signal is sent by the IOU through the central memory port. The most significant bit of the tag field indicates a purge request to P3 when the memory function is equal to "0010" (WRITE). M3 sends a response back to the IOU in exactly the same manner as it would for any other read or write. M3 then sends the purge request to P3.

5.8.2.2.4 Exchange Accept

The exchange accept line is used to indicate that the previous exchange jump request has been honored.

5.8.2.3 General Signals

5.8.2.3.1 Summary Status

The summary status static signal is sent from an external device to the IOU while any bit remains set in the status summary register for that external device. It sets the summary status bit in the IOU Status Summary Register.

5.9 PERFORMANCE MONITORING

Performance monitoring in the IOU is accomplished by means of a combination of hardware and software techniques. Hardware oriented data can be gathered from test points on circuit paks with an external hardware monitor. Software oriented data can be supplied by the PP programs themselves.

5.9.1 Test Points Provided for Performance Monitoring

5.9.1.1 Channel Activity

Test points are provided on the channel circuit paks to allow monitoring of the four channel status signals (active, full, function and flag).

5.9.1.2 PP Program Activity

A test point is provided that allows the sensing of the execution of the 0027 instruction. This allows a keypoint monitoring of particular PP programs.

5.9.1.3 PP to Central Memory Activity

Test points are provided to allow the monitoring of the following PP to Central Memory activity:

PP requests to central memory
Type of request
PP requests that are blocked from access to central memory

6.0 MAINTENANCE CHANNEL

Systems specified in Table 6.0-1 implemented the Maintenance Processor by utilizing a PP in the IOU that has been programmed to act as the Maintenance Control Unit (MCU). The MCU uses the Maintenance Channel Subsystem to access each system element. This Subsystem consists of the Maintenance Channel Interface that is permanently connected to IOU channel 17B, a Maintenance Access Control (MAC) located in each system element, and a set of interconnecting maintenance channels.

The Maintenance Channel Interface contains a selector that specifies one of up to seven maintenance channels for data transmission.

MCI Port	S1	S2	S3/Theta
Port 0	Processor 0, IOU and Memory	IOU	IOU
Port 1	Processor 1*	Memory Unit 0	Processor 0 and Memory Unit 0
Port 2	Unassigned	Processor 0	Unassigned
Port 3	Unassigned	Processor 1*	Processor 1*
Port 4		Unassigned	Unassigned
Port 5		Memory Unit 1*	Memory Unit 1* (Unassigned on THETA)
Port 6		Unassigned	Unassigned

*if present

Table 6.0-1. Maintenance Channel Interface

The maintenance registers supported by each of these systems are summarized in tables 6.0-2 thru 6.0-4. Also each table indicates the intended usage of these registers. Detailed information on how these registers are used can be gained by referencing the appropriate software documents. Registers marked with an asterisk (*) are model-dependent registers in future system designs. Refer to section 9.0 of this document for model-independent requirements on the information that was included in these now model-dependent registers.

REGISTER	System Usage				
	NOS	NOS/VE	DFT	CTI	DIAG
JPS		X	X	X	X
MPS		X	X	X	X
PTA		X	X	X	X
PTL		X	X	X	X
PSM		X	X	X	X
EID		X	X	X	X
SIT		X			X
PID		X			X
PTM*					X
PFS*			X	X	X
DEC*			X	X	X
VMCL		X	X		X
SS*			X	X	X
OI*			X	X	X
KBP		X			X

*Reference section 2.5.1 for model-independent information

Table 6.0-2. Processor State Registers

REGISTER	System Usage				
	NOS	NOS/VE	DFT	CTI	DIAG
SS*			X	X	X
EID			X	X	X
OI*			X	X	X
EC*			X	X	X
BR				X	X
CEL*			X	X	X
UEL*			X	X	X

*Reference section 4.5 for model-independent information

Table 6.0-3. Central Memory Maintenance Registers

REGISTER	NOS	NOS/VE	DFT	CTI	DIAG
SS*			X	X	X
EID			X	X	X
OI*			X	X	X
FS*			X	X	X
FSM*			X	X	X
EC*			X	X	X
TM*					X
OSB†	X		X	X	X
SR*			X	X	X

*Reference section 5.6.2 for model-independent information
†SO does not support OSB

Table 6.0-4. IOU Maintenance Registers

The protocol required to use the Maintenance Access is model-dependent as is the hardware implementation itself. Details on these system implementations can be found by referencing the appropriate equipment specifications.

Changes in the system testability architecture have a dramatic impact on the maintenance access to the system. In order to accommodate a new architecture, the terminology has been changed as indicated in figure 6.0-1. Changes in the maintenance architecture may result in different versions of the supporting software (DFT, CTI, and Diagnostics). Care should be taken in project initiation to minimize the amount of software impact whenever possible.

The Maintenance Processor provides all of the system support functions for each element in the system via a Maintenance Access. These system functions include:

- Initialization
- Error monitoring and analysis
- Reconfiguration
- Verification of hardware

A Maintenance Processor then communicates the results of the execution of these functions to the appropriate higher level software via a Central Memory buffer. (For Operating System software, see DFT/OS Interface Specification, ARH6853.)

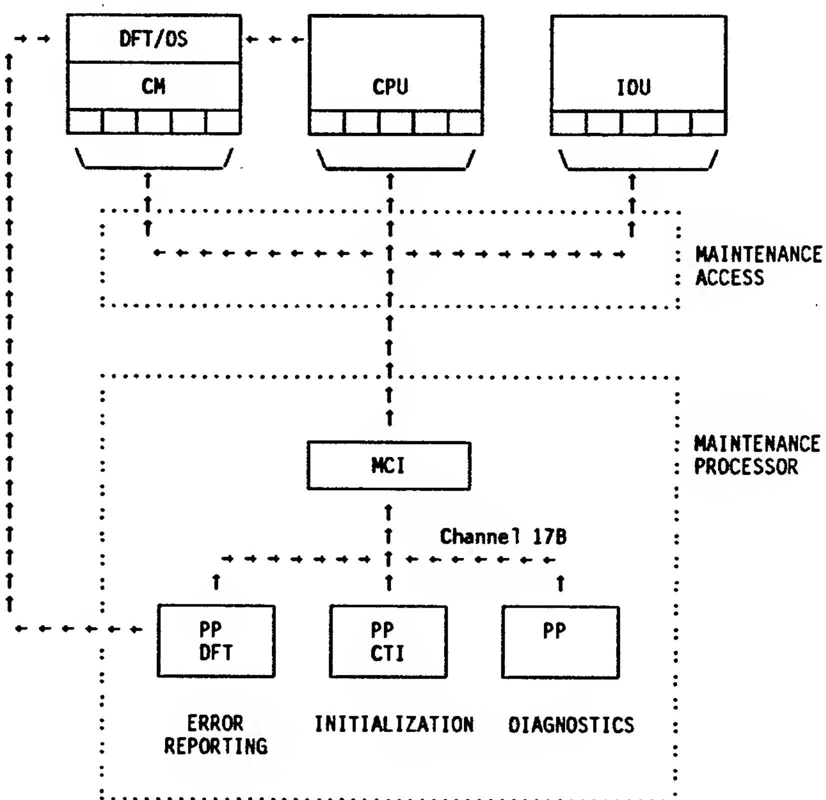


Figure 6.0-1. CYBER 180 Maintenance Architecture

6.1 NOS, NOS/BE SUPPORT

Systems supporting either the NOS or NOS/BE operating systems (C170 mode) must provide the following capabilities:

a) Any connect code from 8 to F will deselect the Maintenance Channel Interface from IOU channel 17B, allowing the IOU to use channel 17B for inter-PP communications with no time-out restrictions.

b) So that nondedicated DFT (PP) can continue to be supported, a single mainframe error status bit shall be accessible in the maintenance register. This status bit shall indicate an error is present anywhere in the configuration. (To maintain compatibility with existing systems, this bit should be located in the IOU Status Summary Register bit 59.)

6.2 MAINTENANCE PROCESSOR FEATURES

As shown in figure 6.0-1, the Maintenance Processor shall have access to all elements of the system. If power/cooling for any mainframe element is a separate external element, then it also must be accessible by the Maintenance Processor. Access should allow the element to concurrently perform its normal operations while the Maintenance Processor is able to determine the element's status per the model-independent definition outlined in section 9.1.1.

The Maintenance Processor shall also have read and write access to Central Memory so that information obtained by the Maintenance Processor can be communicated to higher level software. The actual structure and protocol for this communication is model dependent.

For each element's Maintenance Access the Maintenance Processor shall have the capabilities listed below, if applicable.

- Stop/Start
- Exchange
- Read/Write

6.2.1 Stop/Start Capabilities

A processor shall cease instruction execution and indicate Processor Halt status when either of the following occurs:

- Internally initiated halt

  A condition is encountered that requires a halt as specified in tables 2.8-1 or 2.8-2. The P address shall match the descriptions given in paragraphs 2.8.1 and 2.8.3. The MCR/UCR shall indicate the condition(s) that caused the halt.

- Externally initiated halt

  A STOP PROCESSOR EXECUTION signal is received from the Maintenance Processor. The processor shall halt in an orderly manner such that the halted process may be restarted with integrity; the P address shall point to the next instruction to be executed, etc. This halt need not occur until after the current instruction, trap or exchange is complete. The MCR/UCR will not contain a bit requiring a halt when the halt was externally initiated. Note that a processor having received a STOP PROCESSOR EXECUTION and which encounters an exception condition requiring a halt will set the appropriate bit in the MCR/UCR and then halt matching internally initiated halt rather than the externally initiated halt.

The reason for the halt shall be distinguishable. Halts due to either of the above situations shall be at an instruction boundary, (except for a Detected Uncorrectable Error which may cause a halt anywhere). Error data residing in the processor shall reflect the error status up to the time of the halt. The processor status summary halt bit shall not be set until the processor has halted and is ready for the next maintenance access.

A halted processor may be restarted by either of the following capabilities:

- RESUME - This capability is initiated by the Maintenance Processor. A START PROCESSOR EXECUTION command shall clear the processor holt status and then resume execution of the suspended process without alteration. When a START PROCESSOR EXECUTION signal is received to start processor execution for a processor which has a bit set in MCR/UCR requiring a halt, the processor shall in effect remain halted. This condition can arise when a PP initiates action to read PFS following a corrected error in the processor. If the processor were to encounter a halt due to a bit set in MCR or UCR before the STOP PROCESSOR EXECUTION signal were acted upon, the PP would have no way to recognize that the processor halted due to the processor action rather than the Maintenance Processor action other than the examination of MCR/UCR contents which would not be part of the normal error logging procedure.
- EXCHANGE - This capability places the process information into an exchange package (half-exchange-out), allows reading/writing of the processor and maintenance registers, and then performs a half-exchange-in to execute either the same or a different process. (A half-exchange-out consists of storing the interrupted process into an exchange package in central memory. A half-exchange-in consists of loading a process into the processor from an exchange package in central memory).

The Maintenance Processor shall be able to issue a STOP PROCESSOR EXECUTION, test for the halt, perform any of the Read/Write functions and then restart the halted process by issuing a START PROCESSOR EXECUTION without damaging the process. Reference the appropriate processor engineering specification to provide a detailed description of accessible maintenance registers and any restrictions on writing them.

Notes:

1. The restart of the "halted processor without damage to the process "shall include the integrity of inter-element communications and asynchronous events for the halted processor such as:

C170 Exchange Request	Short Warning
External Interrupt	PIT/SIT
Central Memory Communication	

   When the halted process is C170 Monitor State, the processor is free to send the Exchange Accept either during or at the end of the time period during which the processor is halted. Note that the Short Warning will not cause any processor action until the processor is restarted. The processor shall respond to asynchronous events either between the START PROCESSOR EXECUTION and the first instruction OR between the first and second instruction.

2. The PIT may but need not be stopped by the STOP PROCESSOR EXECUTION function. If the PIT times out while the processor is halted, the bit shall be set in the UCR and appropriate action taken after the START PROCESSOR EXECUTION.

3. The SIT may but need not be stopped by the STOP PROCESSOR EXECUTION function. If the SIT times out while the processor is halted, the bit shall be set in the MCR and appropriate action taken after the START PROCESSOR EXECUTION.

6.2.2 Exchange

The Maintenance Processor shall be able to issue a STOP PROCESSOR EXECUTION, test for the halt and then initiate any or all of the following in the following order:

A. Half Exchange Out

This causes the processor to perform one half of a full exchange by storing the halted process as a C180 Exchange Package into central memory at JPS or MPS as per the C180 monitor flag. The processor then halts and waits for further direction. The monitor flag shall not be altered. Although not a requirement for S0, S1, S2, S3, Theta or PIP3, future systems may simulate this capability within the Maintenance Processor (SCAN-OUT) if they can provide accurate read access to all Exchange Package information.

B. Read/Write Processor Register

The Maintenance Processor may read or write any register in table 2.6-1. See note below.

C. Half Exchange In

This causes the processor to perform one half of a full exchange by loading the process contained in the C180 Exchange Package from JPS or MPS as per the C180 monitor flag. Although not a requirement for S0, S1, S2, S3, Theta or PIP3, future systems may simulate this capability within the Maintenance Processor (SCAN-IN) if they can provide accurate write access to all Exchange Package information.

The sequence of Maintenance Channel operations required to provide any or all of the operations above are model <u>dependent</u>. Providing these capabilities is a model independent requirement.

Notes:

1. The exchange package contents obtained via the half-exchange-out are undefined when any PP write to registers represented in the exchange package occurred before the half-exchange-out.
2. The registers in the processor whose contents are represented in the exchange package following the half-exchange-out shall be specified in the model-dependent engineering specification.
3. The half-exchange-out will be undefined when using an MPS or JPS which has been altered since the last exchange operation.
4. The half-exchange-out data shall specify the complete process state which will allow properly restarting the halted process with integrity.
5. The Maintenance Processor may restart the processor in monitor mode or in job mode. The restart in job mode need only be available when the processor halts in job mode, i.e., there is no requirement for a Maintenance Processor to be able to force a processor halted in monitor mode to initiate a process in job mode via a half-exchange-in. However, it must be possible to force a processor which halted in job mode to initiate a process in monitor mode via a half-exchange-in.
6. The exception testing on the half-exchange-in shall be equivalent to any C180 exchange; i.e., testing for UVMID and DUE.

7. The occurrence of any asynchronous interrupts shall be detected and the appropriate bit set in the MCR for either the exchange package stored into memory on the half-exchange-out or the process initiated via the half-exchange-in. (It is a software responsibility to examine the exchange package produced on the half-exchange-out for any MCR/UCR bits which represent asynchronous events which require software action. It is a hardware responsibility to ensure that all asynchronous events are recognized and set in the MCR for one of the two exchange packages).
8. The PIT may but need not be stopped by the STOP PROCESSOR EXECUTION function. If the PIT times out while the processor is halted and no half-exchange-out has been performed, the bit shall be set in the UCR. The processor shall not allow UCR51 to be set for the new process (via half-exchange-in) because of the PIT which was exchanged out. If the PIT for the new process exchanged in times out, then the bit shall be set in UCR and appropriate action taken after the START PROCESSOR execution.
9. The SIT may but need not be stopped by the STOP PROCESSOR EXECUTION function. If the SIT times out while the processor is halted, the appropriate bit is the MCR shall be set either for the process stored via the half-exchange-out or the new process loaded via the half-exchange-in.
10. The capability to purge cache and MAP via the maintenance access must be provided either as a separate action or as part of the half-exchange-in.

6.2.3 Read/Write Functions

The Maintenance Processor shall be capable of read or write operations on the Maintenance Access to all model-independent maintenance information and the processor registers defined in table 2.6-1, CM registers in table 4.5-1 and IOU registers in table 5.6-3.

Read or Write operations shall be documented in the appropriate model-dependent engineering specification. This description shall include:

Description of format and protocol
Definition of word length
Reaction to the following end conditions, where applicable:
- partial word read or write capability
- access to nonexistent address
- write of read only access
- attempt to read or write more data than required
- protocol or format violations

The Maintenance Processor shall be capable of read or write operations to Central Memory. Read or write operations shall be documented in the appropriate model-dependent engineering specification.

The Maintenance Processor shall be able to read all error information required to make a determination of the Process Damaged/Not Damaged state after a Stop on Error interrupt has occurred. It is the Process Not Damaged assessment that allows the Maintenance Processor to perform an operation retry, take advantage of any degrade or redundancy capability, and recover a task.

Note: Half Exchange-Out should not be used to capture the CPU error data, as it relies on the correct operation of a failing CPU to get accurate information.

A Maintenance Processor shall provide a CLEAR ERROR mechanism which may be used to initialize the maintenance registers and to reset them after the appropriate maintenance information is successfully captured following a detected error condition. The CLEAR ERROR function shall set all maintenance registers to their null state indicating no errors. Details on CLEAR ERROR operation shall be defined in the appropriate model-dependent engineering specification.

Master Clear of an element shall set that element to a defined state. It shall not alter any maintenance registers, processor state registers, or any process state registers.

6.3 OPERATIONS

6.3.1 Initialization

The Maintenance Processor shall provide the ability to initialize the C180 system into a common state prior to the operating system taking control. The detailed definition of this system state shall be defined in the Common Test and Initialization (CTI) Interface Specification (ARH2948). The procedure required to implement this state is model dependent and shall be defined in the appropriate engineering specification. However, each design should accommodate the CTI initialization requirements as part of the system Master Clear design implementation whenever possible. The objective is to minimize the system initialization time.

6.3.2 Monitor and Report Errors

The Maintenance Processor shall provide the interface between the mainframe hardware error reporting mechanism and the operating system. The model-independent characteristics of the hardware to Maintenance Processor Interface are specified in section 9.0. The Dedicated Fault Tolerant to Operating System (DFT/OS) Interface Specification (ARH6853) defines the interface between the Maintenance Processor and the Operating System. This interface shall exist in Central Memory. The unique error reporting information and the operations of the Maintenance Processor shall be defined in the appropriate engineering specifications.

6.3.3 Fault Tolerance

The Maintenance Processor shall provide the interface between the mainframe hardware and the operating system for dynamic reconfiguration of the hardware in response to a detected error condition. This capability is model dependent since it is a direct result of the architectural design of each model.

6.3.4 Diagnostics

Off-line diagnostics may utilize the mainframe element in an abnormal state or partial configuration in order to exploit the hardware design in an effort to create the highest stress condition or improve the diagnostic's fault isolation capability. As a result, off-line diagnostics shall provide their own interface software within the Maintenance Processor.

On-line diagnostics, that execute concurrently with the operating system, shall conform to the model-independent characteristics defined in the DFT/OS Interface Specification.

6.3.5 Fault Injection

The system fault injection capability defined for the mainframe in section 8.0 shall be supported by the Maintenance Processor. The Maintenance Processor shall provide the software interface to the fault injection capability.

6.4 RAM

A fault occurance in the Maintenance Processor of the Maintenance Access shall not affect the integrity of the user operations. Fault tolerant techniques shall be applied to detect errors and isolate faults that will develop in the Maintenance Processor. See section 8.0 for the mainframe fault tolerant design guidelines.

7.0 CYBER 170 STATE

CYBER 170 State shall provide an environment within which a user may execute programs which are comprised of CYBER 170 instructions and which use CYBER 170 data formats. This state is not intended to support unmodified software which is sensitive to small changes in hardware speed. See section 1.6 for a list of systems supporting C170 state operations.

The C170 State is defined only when the C180 Monitor Flag is clear. This fact is presumed throughout this specification and any attempt to initiate a C170 environment with the C180 Monitor Flag set is undefined.

Section 7.1 through 7.13 describe a C170 environment intended to support execution in a single processor. This mode of operation, called solo C170, will not support simultaneous C170 execution in two processors in the same system. Section 7.14 describes an extension of the C170 environment which will support simultaneous execution in two processors in C170 mode in the same system. This mode is called concurrent C170. See section 1.6 for a list of systems supporting concurrent C170.

There are places within section 7 where it is necessary for clarity to use the CYBER 170 numbering convention rather than, or in addition to, the convention described in 2.1.3.6 e/f. At each of these points, the expression C170 bit number X or CYBER 170 bit number X is used.

7.1 CYBER 180 OPERATING SYSTEM

The CYBER 180 Operating System establishes the environment for CYBER 170 State, and provides recovery facilities for hardware and software errors (see 7.6). The minimum capabilities of the CYBER 180 Operating System are as follows:

- a. *It builds the Page Table, Segment Table, and Exchange packages for CYBER 170 State.*
- b. *It exchanges to CYBER 170 State processes (or calls or returns to a CYBER 170 State procedure).*
- c. *It analyzes the reasons for any exchange made back to CYBER 180 State (Job Timer, Page Fault, hardware error, etc.) and takes appropriate action.*

7.2 CYBER 170 STATE MEMORY

7.2.1 Word Format

The 60-bit CYBER 170 words shall be mapped right-justified into 64-bit CYBER 180 central memory words: CYBER 170 bit 59 shall be mapped into CYBER 180 bit 4 and CYBER 170 bit 0 shall be mapped into CYBER 180 bit 63. CYBER 180 bits 0-3 shall be undefined.

C180 Bit Number	0	1	2	3	4	5		62	63
C170 Bit Number	...Undefined...				59	58		1	0

7.2.2 RAC, FLC, RAE and FLE

C170 registers, RAC, FLC, RAE and FLE are contained in the 32-bit BN portion of specific C180 A registers as shown in figures 7.4-1 and 7.4-2 for the C180 Exchange Package and Stack Frame Save Area. These registers shall be stored into the C170 Exchange Package as shown in figure 7.5-1 and shall be limited in size as specified in paragraphs 7.4.2.3, 7.4.2.5, and 7.5.4.

7.2.3 C170 P Register

The C170 P register shall be 18 bits (7.5.1, 7.6.3). The sum of this C170 P register plus RAC (21 bits) shall be contained in the C180 P register as described in 7.4.2.1 and 7.6.4. Thus, it is possible for the processor to allow the C170 P to increment beyond 18 bits; however, the processor operation in C170 State is defined only when:

$$C170\,P < 777{,}777_8$$

When storing P+RAC into an Exchange Package or SFSA, the processor shall ensure that P+RAC is greater than or equal to RAC. Likewise, when producing an Exchange Package or SFSA, the software is responsible for producing P+RAC which is greater than or equal to RAC. Thus, the processor may assume that the P+RAC obtained from the C180 Exchange Package or Stack Frame Save Area when initiating a C170 environment is greater or equal to the RAC obtained from the same Exchange Package or Stack Frame Save Area.

7.2.4 CYBER 170 Memory Facilities

The CYBER 170 State within the CYBER 180 shall have provisions for the following types of memories, as illustrated in figure 7.2-1.

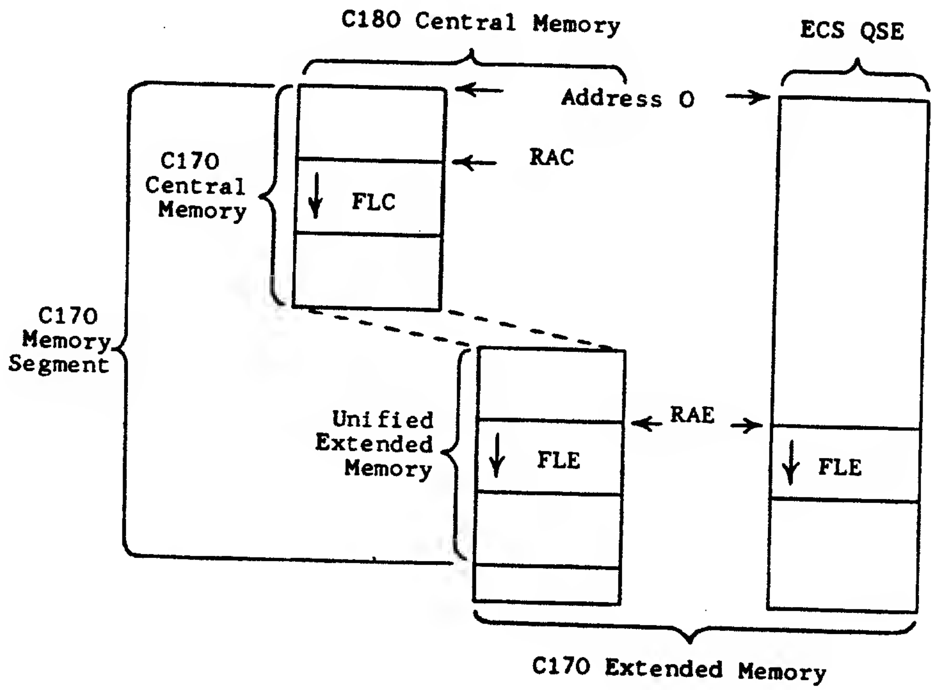


Figure 7.2-1. CYBER 170 Memories

The areas in Central Memory defined by RAC/FLC and RAE/FLE may overlap. If they do, however, this affects the results of the block copy instructions, as described in 7.3.4.

7.2.4.1 C170 Central Memory (CM)

The C170 State Central Memory is that executable portion of the C170 Memory Image Segment that is addressable via RAC and FLC. The processor operation in C170 State is defined only when:

$RAC+FLC < 10{,}000{,}000_8$ $(2{,}097{,}152_{10}$ *words)*

7.2.4.2 Extended Memory

The C170 State extended memory consists of those classes of memory whose address space is defined by the parameters RAE and FLE. These include the following:

- *Unified Extended Memory (UEM)*
  - *UEM (ECS mode)*
  - *UEM (EMS mode)*
- *Extended Core Storage (ECS)**
- *Extended Semiconductor Memory (ESM)**

Unified Extended Memory may operate in two modes, one analogous to ECS and the other analogous to ESM. The C170 State of C180 shall always have Central Memory and Unified Extended Memory (ECS Mode). Supported configurations are listed in table 7.2-1.

	S1 (815,825)	S1CR (810,830)	S2 (835)	S3	THETA (990)
UEM (ECS mode)	Yes	Yes	Yes	Yes	Yes
UEM (ESM mode)	No	Yes	No	Yes	No
ECS	No	No	QSE*	QSE*	No
ESM (ECS mode)	No	No	QSE*	QSE*	No
ESM (ESM mode)	No	No	No	No	No

Table 7.2-1. C170 State Extended Memory

* *Neither a physically separate ECS nor ESM in ECS mode is currently supported by the C170 State of C180. This interface is defined in this chapter to allow future implementation as a QSE if required. However, no plans currently exist to develop this interface.*

The selection of extended memory is controlled by three flags in the C170 State.

ECS Authorized (EA)	*word 2, bit 4*	*C180 exchange package*
UEM Enable	*word 4, bit 23*	
ESM Mode	*word 4, bit 24*	

These flags shall be interpreted as shown in table 7.2-2. See also figures 7.3-1 and 7.3-2, and table 7.4-1.

Flag			Result of 011, 012, 014 or 015 instruction execution:	Treated as an illegal instruction if
ECS Authorized	UEM Enable	ESM Mode		
0	0	0	Illegal instruction	(always)
1	0	0	ECS reference*	ECS not installed*
X	0	1	Illegal instruction	(always)
X	1	0	UEM (ECS mode) ref.	(never)
X	1	1	UEM (ESM mode) ref.	UEM (ESM mode) not installed

* *See 7.2.4.2.1*

Table 7.2-2. Extended Memory Flags

7.2.4.2.1 Extended Core Storage (ECS)

The Extended Core Storage is an equipment external to the C180 processor and central memory.

An interface to ECS, however, does not now exist. Should this interface be required in the future it will be developed as a QSE/QSS rather than as a standard option. Such a QSE would access ECS via the ECS Coupler (7.13) using the 011/012 Block Copy instructions or the 014/015 Direct Read/Write instructions.

7.2.4.2.2 Extended Semiconductor Memory (ESM)

The Extended Semiconductor Memory is an equipment external to the C180 processor and central memory. It is available only as an optional replacement for ECS. ESM is software compatible with ECS except for the following additional ESM features:

- *Additional maintenance features are available via the highspeed port. (See table 7.3-1, 01X Read or Write.)*
- *Additional Flag Register features are available. (See ESM Specification 91915140.)*
- *A "side-door" maintenance facility is available via a channel from the IOU. This port does not have a pass-on, and shall be supported in software only to the extent described in the AO/R (ARH1688).*

7.2.4.2.3 Unified Extended Memory (ECS Mode)

That portion of the C180 Memory Image Segment accessed by RAE and FLE in a manner analogous to accessing ECS in a C170 system is called the ECS mode of Unified Extended Memory - UEM (ECS mode). UEM (ECS mode) shall always be present in C170 State for all systems. The processor operation in C170 State is defined for UEM (ECS mode) only when RAE and FLE fall within the ranges defined in 7.4.2.5. The processor may assume that no word address for UEM (ECS mode) will be encountered which is greater than or equal to $10,000,000_8$ ($2,097,152_{10}$ words) except for fake read (or write) references for the block copy instructions as described in 7.3.4.2. In effect, this requires the software to allow only secure code environments to use fake read, and limit all other environments to be such that

$$RAE+FLE \leq 10,000,000_8 \ (2,097,152_{10} \text{ words})$$

See table 7.3-2, 000 Read or Write.

UEM (ECS mode) may be accessed using the C170 011 or 012 Block Copy instructions, or the 014 or 015 Direct Read/Write instructions.

7.2.4.2.4 Unified Extended Memory (ESM Mode)

That portion of the C170 Memory Image Segment accessed by RAE and FLE in a manner analogous to accessing ESM in a C170 system is called ESM mode of Unified Extended Memory - UEM (ESM mode). UEM (ESM mode) is a standard feature on all S3 systems, but it is not supported in any other C180 configurations. The processor operation in C170 State is defined for UEM (ESM mode) only when RAE and FLE fall within the ranges defined in 7.4.2.5. The processor may assume that no address for UEM (ESM mode) will be encountered which is greater than or equal to $2,000,000,000_8$ ($268,435,556_{10}$) except for fake read (or write) references for the block copy instructions as described in 7.3.4.3. In effect, this requires the software to allow only secure code environments to use fake read, and limit all other environments to be such that

$$RAE+FLE < 2,000,000,000_8 \ (268,435,556_{10} \text{ words})$$

See table 7.3-3, 0 Read or Write.

UEM (ESM mode) may be accessed using the C170 011 or 012 Block Copy instructions, or the 014 or 015 Direct Read/Write instructions.

7.2.5 CYBER 170 Memory Image Segment

The entire CYBER 170 State memory image, including C170 CM and UEM shall be addressed as a single CYBER 180 segment.

The C170 word address after RAC, RAE addition in C170 State is left-shifted three places (with zero insertion) to become the BN of the PVA for the segment containing the C170 memory image.

7.2.5.1 P Ring/Segment Number

The segment number and ring number from the current C180 P register is used for all processor generated memory references while in C170 State. These include:

- *Instruction Fetch*
- *Load/Store*
- *UEM transfer*
- *Conversion of MA to a PVA on CEJ and MAN Exchanges*
- *Conversion of Bj+K to a PVA on CEJ Exchanges*
- *Conversion of PP supplied Exchange address to PVA on MXN and EXN Exchanges*

During conversion, the 18-bit exchange addresses are shifted left three bit positions into the 32-bit BN of the PVA, with zero insertion on both the right and the left.

The only C180 State Processor generated addresses not treated as a PVA are those transmitted to the ECS Coupler (when the ECS QSE is present) which are in turn converted to RMAs and used to address central memory.

Further, the ring number and segment number for the C180 P register are derived from the same source as the P byte number upon entering C170 State and they do not change while within C170 State.

The C170 P register contents come from:

- *The C180 Exchange Package when exchanging into C170 State.*
- *The Code Base Pointer when entering C170 State via a Call. (P ring number generated as described in paragraph 2.6.1.2.)*
- *The Stack Frame Save Area when entering C170 State via a Return.*

7.2.5.2 Page Table Search Without Find (Page Fault)

Page Faults may occur within a C170 environment on any processor generated reference (see list in previous paragraph). Those page faults not the result of an address out of range as described in 7.6.4 shall set MCR57 and place the address (including ring and segment number) in the Untranslatable Pointer Register. When the setting of MCR57 results in a C180 Exchange, the C170 environment is restartable only when the page fault resulted from a reference to UEM using the 011, 012, 014, 015 instructions. The C170 environment is not required to be restartable following a page fault resulting from any other reference.

When the Page Fault does not occur on the initial word transferred, the 011, 012 instructions may or may not transfer some data before the Page Fault is detected. In either event, the entire transfer will be restarted from the beginning when the C170 environment is reinitiated.

7.2.5.3 Address Spec Error, Invalid Segment, Access Violation

These conditions are tested on each memory reference in the C170 State. This C170 State task need not be restartable following these faults.

C170 instruction execution requires the C170 segment to have execute privilege (01, 10 or 11 in XP field of segment descriptor), and P.RN in the execute bracket. Memory write operations (CM or UEM) require write access, appropriate key/locks if selected, and P.RN $\leq$ R1. Memory read operations (CM or UEM) require read access, appropriate key/locks if selected, and P.RN $\leq$ R2. A C170 exchange requires the same privilege as a read and a write operation.

7.2.5.4 Cache Purge

The processor in C170 State shall reference the Segment Table and Page Table and shall purge cache on processor stores into Central Memory just as in C180 State.

There shall be provision to purge cache for IOU 60-bit store operations into the lowest 16 MB of Central Memory. The RMA from the IOU shall be catenated with an ASID of $FFFF_{16}$ and this SVA shall then be used to purge any matching entry in cache. (Note that the intended use of this is for pages mapped 1:1, BN=RMA as described in the following paragraph.)

The ASID value of $FFFF_{16}$ shall be globally reserved to support this cache invalidation hardware.

See section 7.14 and 7.14.4 for description of addition purges required to support concurrent C170 environments.

7.2.5.5 Mapping

All processor generated memory references in C170 State are PVA's and are translated via the mapping inherent in the Page Table (with the single exception of addresses (PVA's) which are sent to the ECS Coupler and are directly converted to RMA's-BN from PVA used as RMA). Thus, pages of a C170 environment need not be mapped 1:1 except as required to facilitate PP interaction (for example, cache invalidation on I/O Writes) or to facilitate the use of ECS.

The following two paragraphs indicate how these features might be used. Note that the hardware performs the same in both cases.

- *Mapping 1:1 A170 NOS, NOS/BE*

  This mapping of the segment utilizes the processor cache invalidation hardware and requires the following:

  a. *The Page Table shall be set up by software such that the CYBER 170 memory image addresses map 1:1 to real memory; i.e., the RMA obtained from the Page Table shall equal BN from the PVA when in CYBER 170 State.*

  b. *The Segment Table shall be set up by software such that the ASID obtained in CYBER 170 State shall be FFFF.*

  c. *The processor hardware shall invalidate those entries in the cache whose ASID is equal to FFFF and whose BN is equal to the RMA supplied by the IOU on a 60-bit write to CYBER 180 central memory.*

  d. *Any 64-bit writes to central memory by the IOU do not purge the cache, and thus, can cause erroneous results if referencing CYBER 170 State memory.*

- *Other Mapping (not 1:1)*

  Another mapping for the CYBER 170 memory image is possible using software cache invalidation. For this case the following would be implemented:

  a. *The page table could be set up by software such that the CYBER 170 Memory image segment (other than perhaps portions of central memory involved in transfers with physical ECS) need not map 1:1 to real memory addresses. Any PP or ECS Coupler interaction with C170 memory would require special attention by software.*

  b. *The management of cache purging in support of I/O operations would be done by software.*

  c. *The software could generate IOU central memory write instructions for 64-bit words because an IOU 60-bit write would activate the same cache invalidation hardware as previously described, thus, causing unnecessary CPU performance degradation.*

7.3 CENTRAL PROCESSOR INSTRUCTION SET

In CYBER 170 State the central processor shall execute instructions and produce arithmetic results in the absence of error conditions according to CDC-SPEC 19063000 (CYBER 170/173 Engineering Specification), with the exceptions listed below and in 7.6 and 7.7.

7.3.1 Compare/Move Instructions

Every CYBER 170 Compare/Move instruction (op codes 464, 467) shall be handled in one of two ways.

It may be executed as specified in CDC Spec. 19063000. This includes detection of these op codes as illegal instructions when the instruction is not in parcel 0. Interrupts shall be recognized and the instruction interrupted with P pointing to the instruction (as specified in CDC Spec. 19063000). See Error Exit table 7.6-1.

It may be detected as an Unimplemented Instruction, setting bit 49 of the User Condition Register. This results in either a Trap or an Exchange (see table 2.8-2) to the CYBER 180 state environment, with the PVA stored in word zero of the Stack Frame Save Area or in the Exchange Package pointing at the Compare/Move instruction in the CYBER 170 state environment. This is true regardless of the parcel in which the Compare/Move instruction appears. If other than parcel 0, the software is responsible for recognizing it as an Illegal instruction. Refer to GID No. ARH2949, C180 State Interface Software for the A170 Computer.

7.3.2 Trap 180 Instruction

The CYBER 170 Op code 017jk shall be redefined as the TRAP 180 instruction and shall cause the Privileged Instruction Fault bit (UCR48) to be set in the User Condition Register. This will cause an interrupt to occur as defined in table 2.8-2.

7.3.3 Direct Read/Write Central Memory

Two 15-bit CYBER 170 Mode CPU instructions shall be added that permit a CYBER 170 X register to be loaded from or stored to any location in CM. These instructions are:

- *660jk Read CM at (Xk) to Xj*
- *670jk Write Xj into CM at (Xk)*

The rightmost 21 bits of Xk specify the memory address relative to RAC. C170 bits 21 through 29 must be set to zero or the operation of this instruction is undefined. Undefined in this instance means specifically that setting any of bits 21 through 29 may cause an Address Out of Range condition to be reported. When Xk ≥ FLC the instruction shall detect Address Out of Range (see 7.6-1). C170 bits 30 through 59 are ignored.

7.3.4 Block Copy Instructions

- *011jK Block Copy Bj+K words from (X0+RAE) to (A0+RAC)*
- *012jK Block Copy Bj+K words from (A0+RAC) to (X0+RAE)*

The 011 and 012 Block Copy instructions operate in one of three modes selected as described in figure 7.3-1 and the following paragraphs:

ECS	*(7.3.4.1)*	*UEM (ESM mode)*	*(7.3.4.3)*
UEM (ECS mode)	*(7.3.4.2)*		

When the Block Copy Flag (see table 7.4-1) from the Exchange Package is set, the 011 and 012 instructions select X0 Upper (C170 bits 30-50) instead of A0 for addressing Central Memory. In this case, X0 shall be interpreted as follows:

```
                  59   51 50          30 29     23                 0
                 +------+-----------+-----+--+--------------------+
X0 (ECS mode):   | zeros| CM Address|/////|  |    ECS Address     |
                 +------+-----------+-----+--+--------------------+
                                             ↑
                                   ← ← ← ← Flag Register bit
                                        ↓
                  59   51 50          30 29                        0
                 +------+-----------+--+--------------------------+
X0 (ESM mode):   | zeros| CM Address|  |       ESM Address        |
                 +------+-----------+--+--------------------------+
```

C170 bits 51-59 are reserved and must be set to zero.

The corresponding field length testing for CM shall be:

X0(21 bits) + Bj(18 bits) + K(18 bits) ≤ FLC(21 bits)

When the Block Copy Flag is clear in ESM mode, A0 is selected for addressing central memory and C170 bits 30 through 59 are ignored. When the Block Copy Flag is clear in ECS mode, A0 is selected for addressing central memory and C170 bits 24 through 59 are ignored. The corresponding field length testing for CM shall be:

A0(18 bits) + Bj(18 bits) + K(18 bits) ≤ FLC(21 bits)

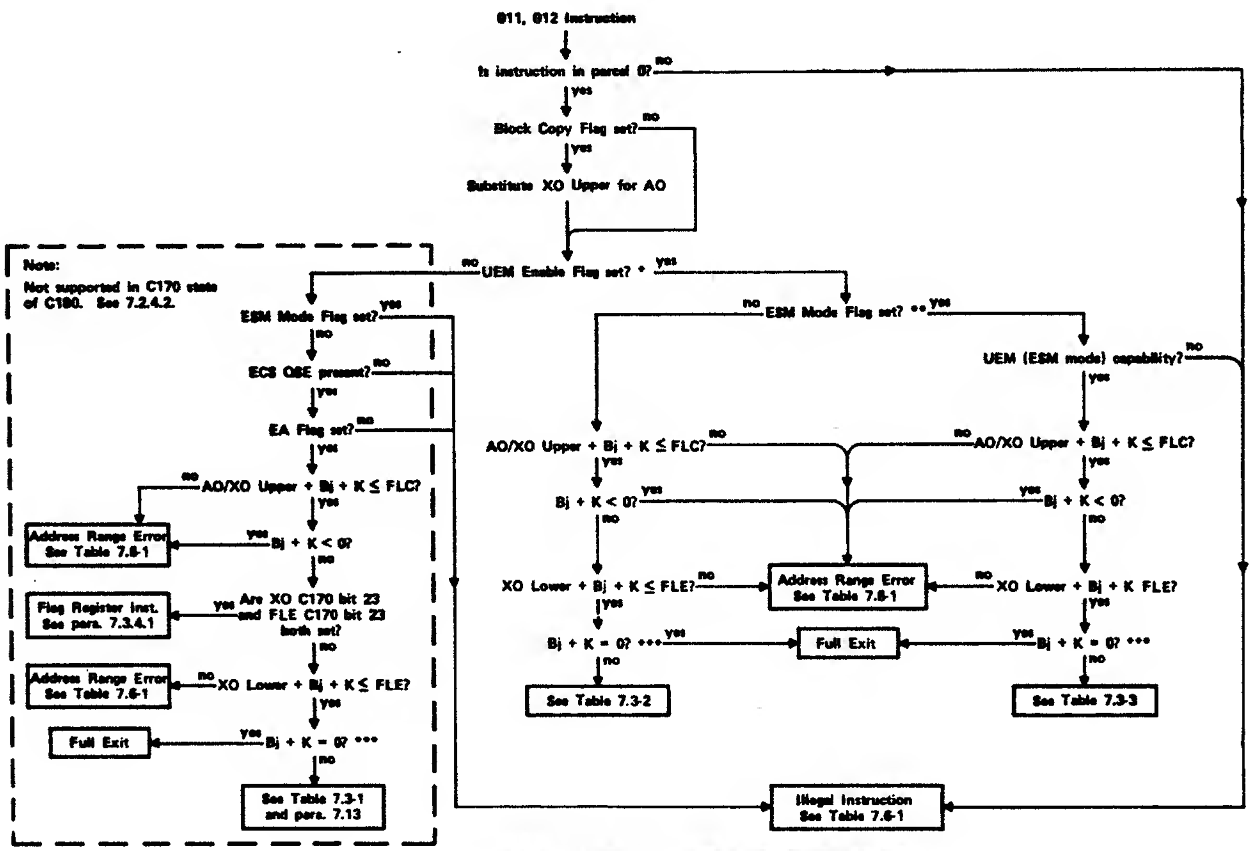
All field length testing for block copy instructions must ensure that the entire block transferred falls within the address fields defined by RAC/FLC or RAE/FLE.

When the input and output fields overlap in physical memory addresses, and the starting address of the output field is larger than the starting address of the input field, the final contents of the output field are undefined.

The following CYBER 170 terms are used in the 011 and 012 instruction descriptions and are defined in the CYBER 170 Hardware Reference Manual and Engineering Specification.

ILLEGAL INSTRUCTION	*FULL EXIT*
HALF EXIT	*ERROR EXIT*

The 011 and 012 instructions shall be implemented as indicated in figure 7.3-1 and tables 7.3-1, 7.3-2, 7.3-3 and the following paragraphs.



*UEM Enable flag test is mandatory for models having both ECS/ESM and UEM. Optional test on other models.

**ESM Mode flag test is mandatory for models having UEM (ESM mode). Optional test on other models.

***Bj + K interpreted as 17 bits plus sign bit for 131K maximum.

Figure 7.3-1. 011, 012 Instructions

7.3.4.1 ECS

These instructions shall be block copy instructions when either bit 23 of X0 or bit 23 of FLE is clear, and shall be Flag Register instructions when bit 23 of X0 and bit 23 of FLE are both set. These instructions shall be software compatible (except where specifically noted) to the 011 and 012 instructions on a CY173 with the AT280 ECS Coupler when referencing ECS. Registers A0, X0 and Bj are not altered by the execution of these instructions.

- *Block Copy*

  As a block copy instruction, 011 reads a block of Bj+K 60-bit words from consecutive addresses that begin at (X0) + RAE in ECS into consecutive addresses that begin at (A0) + RAC in central memory. Likewise, the 012 instruction writes a block of Bj+K 60-bit words into consecutive addresses in ECS.

- *Flag Register*

  As Flag Register instructions, both the 011 and 012 instructions perform a Flag Register operation in ECS. In this case, X0 (rather than X0 + RAE) is transmitted to the ECS Coupler.

  The CYBER 170 State shall perform Flag Register operations as defined in the CYBER 170 ECS Hardware Reference Manual, (60430000) with the following exceptions.

  - *The CYBER 173 performs an ERROR EXIT with condition bit 51 set when a parity error is detected in the transfer of the address/word count from the processor to the ECS Coupler or in the address (function word) from the ECS Coupler to the ECS Controller.*
  - *The CYBER 180 in CYBER 170 State shall do the following when a parity error in the address/word count is detected either by the ECS Coupler or Controller.*

    1. *The ECS Coupler shall transmit an ERROR END OF OPERATION signal to the processor and shall set bit 61 of the ECS Coupler Status Summary register.*
    2. *The ERROR END OF OPERATION from the ECS Coupler shall cause the processor to set the DUE bit in the Monitor Condition Register. (See table 2.8-1.)*

- *4 Million Word ECS QSE*

  The 4 Million Word ECS systems use 22 bits of X0 for the ECS address. In addition, C170 bit 22 when set on an ECS write causes the write to take place in both the upper and lower 2 Million words of ECS.

  The installation of a 4 Million Word ECS QSE will require a change in the processor to delete the test for C170 bit 21 and the subsequent Fake READ.

NOTE: This table must be used in conjunction with figure 7.3-1.

ECS Starting Address C170 bits 23, 22, 21	CYBER 173	CYBER 180 in CYBER 170 State
000 (Read or Write ECS)		
1. ERROR FREE TRANSFER	Complete Entire Transfer	
$Bj + K > 0$ $X0 + Bj + K \leq FLE$ $A0 + Bj + K \leq FLC$	FULL EXIT	
2. ECS Bank not available • maintenance mode • lost power • not in system	No additional data is transferred (including current record) HALF EXIT	
3. Parity error in Address or Word count from Processor to ECS Coupler	No data is transferred HALF EXIT	ECS Coupler shall immediately transmit ERROR END OF OP to Processor - see Note 1. Since the ECS transfer is broken into blocks and the ECS Coupler is initialized before each block, it is very possible that several blocks may have been transferred before the error occurs.
4. Parity error in Address from Processor to central memory controller	Data is transferred to erroneous address FULL EXIT	N/A

Table 7.3-1. ECS Block Copy (Part 1 of 4)

NOTE: This table must be used in conjunction with figure 7.3-1.

<table>
<tr><th>ECS Starting Address
C170 bits 23, 22, 21</th><th>CYBER 173</th><th>CYBER 180 in
CYBER 170 State</th></tr>
<tr><td>000 (Read ECS)</td><td></td><td></td></tr>
<tr><td>1. Parity error in address from ECS Coupler to ECS Controller</td><td>Complete entire transfer with zero data (proper parity) to CM from point of error
HALF EXIT</td><td>Terminate transfer of current block of data. Any additional data blocks may be transferred without error
HALF EXIT</td></tr>
<tr><td>2. Parity Error in address from Central Memory Control (CMC) to Central Storage Unit (CSU)</td><td>Complete entire transfer. No data stored in Central Memory for those words whose associated address had a parity error.
HALF EXIT</td><td>Analogous type of error will result in the ECS Coupler transmitting an ERROR END OF OPERATION to the processor immediately as shown in item 5 below. See Note 1.</td></tr>
<tr><td>3. Parity error in data detected by ECS Controller or ECS Coupler</td><td colspan="2">Complete entire transfer including erroneous data
HALF EXIT
See Note 2.</td></tr>
<tr><td>4. Parity error in data from ECS Coupler detected by Central Memory Controller</td><td>Complete entire transfer including erroneous data
HALF EXIT</td><td>Analogous type of error will send immediate ERROR END OF OPERATION to processor as shown in item 5 below.
See Note 1.</td></tr>
<tr><td>5. Any response from Central Memory on ECS Read other than WRITE RESPONSE
WRITE CORRECTED
ERROR RESPONSE</td><td>N/A</td><td>ECS Coupler shall immediately transmit ERROR END OF OP to Processor.
See Note 1.</td></tr>
<tr><td>6. Read of block of data starting within physically installed ECS but continuing into nonexistent memory</td><td colspan="2">Transfer ECS data until nonexistent address is encountered, then complete transfer with zero data (proper parity) to CM.
HALF EXIT</td></tr>
<tr><td>7. Read of block of data starting above physically installed ECS</td><td colspan="2">Complete entire transfer with zero data (proper parity) to CM.
HALF EXIT</td></tr>
</table>

Table 7.3-1. ECS Block Copy (Part 2 of 4)

NOTE: This table must be used in conjunction with figure 7.3-1.

ECS Starting Address C170 bits 23, 22, 21	CYBER 173	CYBER 180 in CYBER 170 State
000 (Write ECS)		
1. Parity error in address from ECS Coupler to ECS Controller	No additional data is transferred (including current ECS record) HALF EXIT	
2. Parity Error in address from CMC to CSU	Complete entire transfer. All ones data to ECS for words associated with parity error. HALF EXIT	Analogous type of error will send immediate ERROR END OF OPERATION to processor as shown in item 7 below. See Note 1.
3. Corrected error in data read from central memory	Complete entire transfer FULL EXIT	Complete entire transfer ECS Coupler shall cause bit 14 of MCR to be set FULL EXIT
4. Uncorrectable error in data read from central memory	Complete entire transfer including erroneous data HALF EXIT	ECS Coupler shall immediately transmit ERROR END OF OP to processor. See Note 1.
5. Parity error in data from central memory detected at ECS Coupler	Not tested	Complete only current data block including erroneous data. HALF EXIT
6. Parity error in data from ECS Coupler detected by ECS Controller	Complete entire transfer including erroneous data. HALF EXIT	Complete only current data block including erroneous data. HALF EXIT
7. Any response from Central Memory on ECS Write other than READ RESPONSE READ RESPONSE CORRECTED ERROR	N/A	ECS Coupler shall transmit ERROR END OF OP to Processor. See Note 1.
8. Write of block of data starting within physically installed ECS but continuing into nonexistent memory	Transfer data until nonexistent address is encountered and then stop data transfer. HALF EXIT	
9. Write of block of data starting at address above physically installed ECS	No data transfer HALF EXIT	

Table 7.3-1. ECS Block Copy (Part 3 of 4)

NOTE: This table must be used in conjunction with figure 7.3-1.

ECS Starting Address C170 bits 23, 22, 21	CYBER 173	CYBER 180 in CYBER 170 State
001 Read	Complete entire transfer with zero data (proper parity) to CM HALF EXIT See Note 3	
001 Write	No data transfer HALF EXIT See Note 4	
01X Read	ECS Complete entire transfer with zero data (proper parity) to CM HALF EXIT	
	ESM in ECS Mode Complete entire transfer with zero data (proper parity) to CM. Perform maintenance function(s) in ESM. FULL EXIT See Notes 5, 6	ESM in ECS Mode No data transfer. Perform 1 maintenance function in ESM. FULL EXIT See Notes 6, 7
01X Write	ECS No data transfer HALF EXIT	
	ESM in ECS Mode No data transfer. Perform maintenance function(s) in ESM. FULL EXIT See Notes 6, 8	ESM in ECS Mode No data transfer. Perform 1 maintenance function in ESM. FULL EXIT See Notes 6, 7

Table 7.3-1. ECS Block Copy (part 4 of 4)

Notes for table 7.3-1

1. *The ERROR END OF OPERATION from the ECS Coupler shall cause the DUE bit in the Monitor Condition Register to be set. (See table 2.8-1.)*
2. *Parity Error on ECS Read*

   The actions given in the table for data parity errors on ECS Read refer specifically to parity errors on data to be transferred to central memory. If the parity error were on data beyond that required by the word count, the parity error will be ignored. For example, a single word Read of word 4 from ECS Record that has parity errors in words 3 and 5 shall not HALF EXIT, etc.
3. *When executing the 011 instruction, the C180 processor shall detect whether or not C170 bit 21 is set in the ECS address (X0+RAE), and when set shall transfer zeros to central memory without involving the ECS coupler. However, the ECS Coupler shall be able to convert this address into a False Read as explained in 7.3.4.2. The CPU in effect shall handle this as shown in table 7.3-1 001 Read (including the interrupt restrictions in note 1).*
4. *When executing the 012 instruction, the C180 processor may, but need not detect whether or not C170 bit 21 is set in the ECS address (X0+RAE). The processor shall do whatever is most efficient on a model-dependent basis. The instruction shall HALF EXIT with no data transferred whether or not the ECS Coupler is initiated (see 7.3.4.2).*
5. *Transfer Bj+K words consisting of zeros into the CY173 central memory and FULL EXIT (assuming no malfunction occurs which would cause HALF EXIT). The ESM (in ECS mode) will interpret each ECS address sent by the coupler to specify a Maintenance function as defined in the ESM Spec. 91915140. More than one ECS address (function of Bj+K and Starting ECS address) may be sent to ESM (in ECS mode), each of which may set up Maintenance functions.*

   Note that it is also possible for the ECS address to increment such that a 100 or Flag Register code is sent to ESM (in ECS mode).
6. *The Maintenance functions in ESM (in ECS mode) are such that subsequent references to the high speed port are affected; i.e., an "accidental" Maintenance function will typically cause subsequent references to produce erroneous data.*
7. *The CYBER 180 ECS Coupler will detect this code and operate much like a Flag Register Operation in that no data is transferred and only one ECS Starting Address (or Maintenance function) is sent to ESM (in ECS mode). A FULL EXIT will occur at the end of this operation.*

8. *The CYBER 173 will read Bj+K words from its central memory and transmit them to ESM (in ECS mode). The ESM (in ECS mode) will ignore the data and perform a Maintenance function (as defined in the ESM Spec.) for each ECS address received. The CYBER 173 will then FULL EXIT at the end of this operation assuming that no malfunctions (such as SECDED error on central memory read) have occurred. Note that incrementing the ECS address could result in a 100 or Flag Register code being sent to ESM (in ECS mode).*

9. *ESM in ECS Mode*

   The ESM in ECS mode is explicitly covered only in table 7.3-1 where it is known to be different from ECS.

10. *Multiple Failures*

    In the event of multiple failures during a block transfer, the following priority shall be observed relative to the signals from the ECS coupler (see 7.13).

    ERROR END OF OPERATION
    HALF EXIT
    FULL EXIT

11. *There are other conditions in the CYBER 170 State of CYBER 180 such as ECS Coupler Buffer parity error which will produce ERROR END OF OPERATION that have no analogous case in CY173 (see C180 ECS Coupler Spec.).*

12. *Simultaneous Field Length Error and PP Exchange Request*

    On CYBER 173, a simultaneous field length error and PP Exchange Request do not store the exit condition bits at RAC even though the P register has been cleared (see table 5-7 of the CYBER 170 Hardware Reference Manual). The CYBER 175 stores the exit condition bit at RAC as specified. The CYBER 170 State of CYBER 180 will store the exit condition bits for this case rather than track the CYBER 173.

7.3.4.2 UEM (ECS Mode)

As a block copy instruction, the 011 reads a block of Bj+K 60-bit words from consecutive addresses that begin at (X0+RAE) in Unified Memory into consecutive addresses that begin at (A0+RAC) in Central Memory. Likewise, the 012 writes a block of Bj+K 60-bit words into consecutive addresses in UEM.

For the block copy in UEM (ECS mode), the field length testing for UEM shall be:

$$X0(24\ bits) + Bj(18\ bits) + K(18\ bits) \leq FLE(23\ bits).$$

The processor may assume C170 bit 23 of FLE to be zero; if it is not zero, the result of the field length test is undefined.

Note: This table and associated notes must be used in conjunction with figure 7.3-1.

UEM Starting Address (X0+RAE) C170 bits 23, 22, 21	CYBER 180 in CYBER 170 State
000 Read or Write Bj + K > 0 X0 + Bj + K ≤ FLE A0 + Bj + K ≤ FLC	Complete entire transfer FULL EXIT See Note 1
001 Read, OR 01X Read	Fake Read Complete entire transfer with zero data (proper parity) to CM. (See Notes 1 and 5). HALF EXIT
001 Write, OR 01X Write	No data transfer HALF EXIT

Table 7.3-2. UEM (ECS Mode) Block Copy

Notes for tables 7.3-2 and 7.3-3

1. *The 011 and 012 instructions when referencing UEM shall divide transfers greater than 64 words in length into a series of shorter data blocks (not to exceed 64 words each) so as to provide greater interrupt response. The processor shall respond to bits which set in the MCR or UCR by terminating the transfer between two of these data blocks and taking the appropriate action as defined in tables 2.8-1 and 2.8-2 of the MIGDS. When the interrupted program is restarted, the data transfer will be restarted from the beginning.*

   These data blocks shall be chosen such that the transfer will be interrupted only between the equivalent of ECS records and not between the last two records. ECS records begin at ECS word addresses equal to 0, modulo 8; thus, the equivalent records in UEM begin at UEM word addresses equal to 0, modulo 8.

2. *The 011 and 012 instruction exits, HALF or FULL, shall be as defined in tables 7.3-2 and 7.3-3 when referencing UEM in the absence of Uncorrected Errors. Any Corrected Error shall cause the specified exit with the Corrected Error bit set in the MCR (2.7.1.15 of the MIGDS). Any Uncorrected Error shall cause the setting of the Detected Uncorrectable Error bit in the MCR and the program interruption specified in table 2.8-1 of the MIGDS.*

3. *Page Faults, if encountered in UEM, on the 011 or 012 instruction will always occur between the equivalent of ECS records because the page boundaries occur between ECS records. It remains then for the processor to pretest or prevalidate the last ECS record appropriately to assure that the transfer will not be interrupted between the last two records.*

4. *On the CYBER 173, a simultaneous field length error and PP Exchange Request does not store the exit condition bits at RAC even though the P register has been cleared (see table 5-7 of the CYBER 170 Hardware Reference Manual). The CYBER 175 stores the exit condition bit at RAC as specified. The CYBER 170 State of CYBER 180 will store the exit condition bits for this case rather than track the CYBER 173.*

5. *A block transfer whose starting address (X0+RAE) does not have the appropriate Fake Read bits set initially, but whose length is such that these bits set during the transfer, nevertheless completes the entire transfer with FULL exit. The Fake Read or No Data Transfer with HALF exit operations are invoked only when the appropriate bits are set in the starting address.*

6. *Read operations from ECS which start in existent memory but continue into addresses for which no memory exists will be converted into zero-fill transfers on the C173. Read transfers from UEM which start in existent memory as defined in the Page Table but then continue into addresses having no Page Table definition will result in Page Faults rather than zero fill.*

7.3.4.3 UEM (ESM Mode)

As a block copy instruction, the 011 reads a block of Bj+K 60-bit words from consecutive addresses that begin at (X0+RAE) in Unified Extended Memory into consecutive addresses that begin at (A0+RAE) in Central Memory. Likewise, the 012 writes a block of Bj+K 60-bit words into consecutive addresses in UEM.

For the block copy in UEM (ESM mode), the field length testing for UEM shall be:

$$X0(30\ bits) + Bj(18\ bits) + K(18\ bits) \leq FLE(29\ bits).$$

The processor may assume C170 bit 29 to be zero; if it is not zero, the result of the field length test is undefined.

Note: This table and associated notes must be used in conjunction with figure 7.3-1.

UEM Starting Address (X0+RAE) C170 bit 28	CYBER 180 in CYBER 170 State
0 Read or Write Bj + K > 0 X0 + Bj + K ≤ FLE A0 + Bj + K ≤ FLC	Complete entire transfer FULL EXIT See Note 1
1 Read	Fake Read Complete entire transfer with zero data (proper parity) to CM. (See Notes 1 and 5). HALF EXIT
1 Write	No data transfer HALF EXIT

Table 7.3-3. UEM (ESM Mode) Block Copy

The notes for table 7.3-3 are identical to those for table 7.3-2, and are found in paragraph 7.3.4.2.

7.3.5 Direct Read/Write Extended Memory

- *014jk Read one word from (Xk+RAE) to Xj*
- *015jk Write one word from Xj to (Xk+RAE)*

The 014 and 015 Direct Read/Write instructions operate in one of three modes (ECS, UEM [ECS mode], or UEM [ESM mode]) selected as described in paragraph 7.2.3 and figure 7.3-2.

The 014 instruction shall read the 60-bit word from the address in extended memory formed by Xk+RAE and load it into register Xj. The 015 instruction shall write the 60-bit word from register Xj into the address in extended memory formed by Xk+RAE.

Register Xk is not altered by the execution of this instruction.

Field length testing shall be as follows:

- *ECS and UEM (ECS mode)*

  A 24-bit compare of Xk and FLE. The processor may assume C170 bit 23 of FLE to be zero; if it is not zero, the result of the field length test is undefined.

- *UEM (ESM mode)*

  A 30-bit compare of Xk and FLE. The processor may assume C170 bit 29 of FLE to be zero; if it is not zero, the result of the field length test is undefined.

The 014 and 015 instructions do not support Flag Register operations. When the Flag Register bit (C170 bit 23 in ECS mode, C170 bit 29 in ESM mode) of both Xk and FLE is set, these instructions will set Address Out of Range. (See table 7.6-2.)

The 014 and 015 instructions do not perform any special testing in support of fake read. When referencing UEM, there is no fake read (zero fill). When referencing ECS, 014 will fake read and 015 will cause no write whenever the address in Xk (after field length test) requests an address above the installed, available memory size. (See table 7.3-1.)

After appropriate field length testing when referencing UEM, the 014 and 015 instructions will transmit the Byte Number to Central Memory for UEM (ECS mode) this is 24 bits of Xk with zero fill. For UEM (ESM mode), this is 30 bits of Xk with zero fill.

C170 bits 24-59 or Xk are ignored for ECS and UEM (ECS mode). C170 bits 30-59 of Xk are ignored for UEM (ESM mode).

Page Faults, if encountered, in UEM on the 014 or 015 instruction shall cause a program interruption as defined in table 2.8-1 of the MIGDS.

The 014 or 015 instructions exit to the next parcel in the absence of Uncorrected errors. (Any Corrected Error shall cause the normal exit with the Corrected Error bit set in the MCR.) Any Uncorrected Error shall cause the setting of the Detected Uncorrectable Error bit in the MCR and the program interruption specified in table 2.8-1 of the MIGDS.

These instructions shall be executed as indicated in figure 7.3-2.

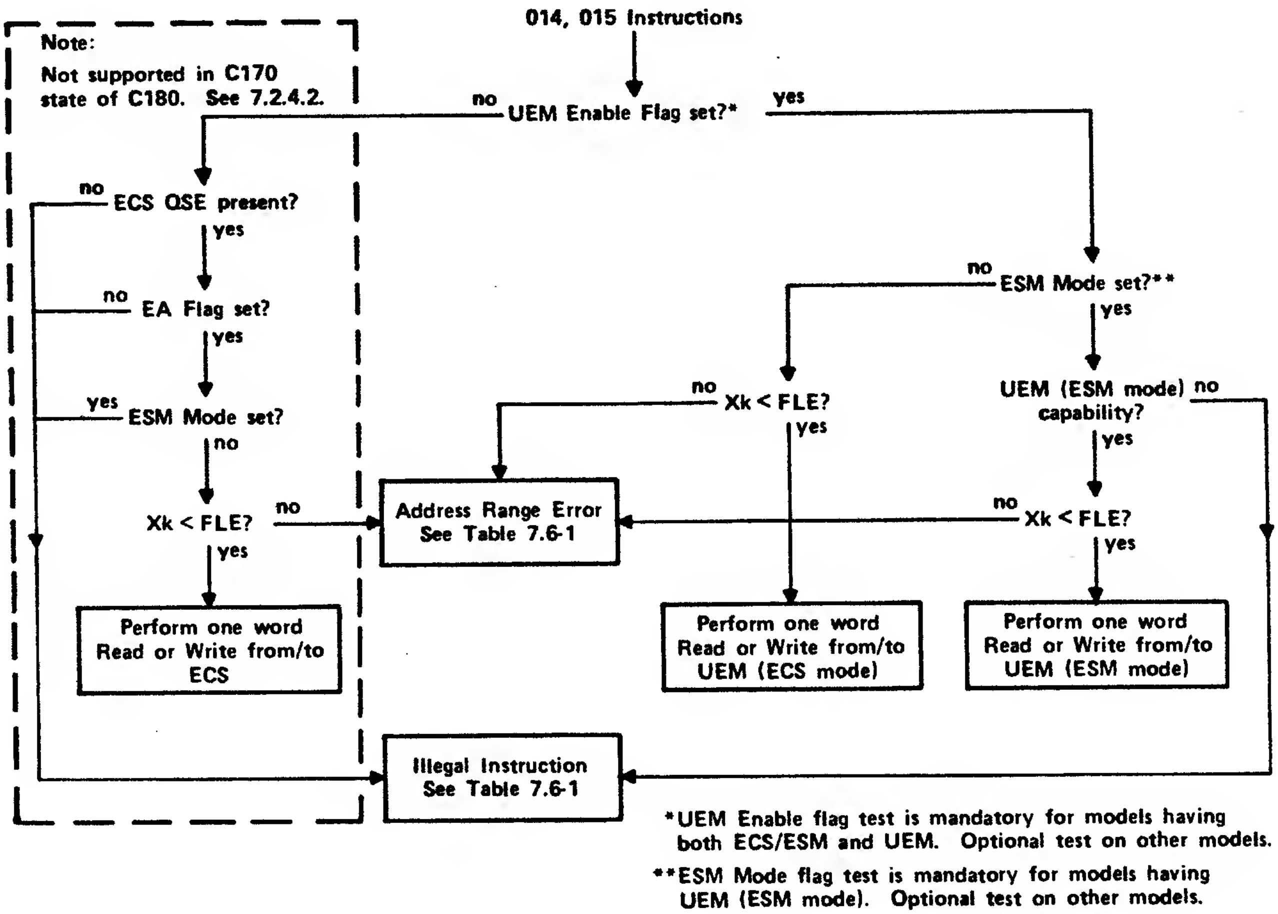


Figure 7.3-2. 014, 015 Instructions

7.3.6 Read Free Running Counter

- *016jk Read Free Running Counter*

This instruction shall transfer the current 48-bit contents of the Free Running Counter (4.2.3.4) into the Xj register. The leftmost twelve bits of Xj shall be cleared to zeros.

The Processor Memory Port to be utilized during the execution of this instruction shall be determined in the manner defined in paragraph 2.10.1.1 of this specification with the exception that, for this instruction, C170 bit 30 of the Xk Register shall be used in place of bit 33 of the Real Memory Address. The remaining bits in Xk shall be ignored by the processor.

This single parcel instruction may be located in any parcel.

7.3.7 Central Exchange Jump to (Bj)+K (C170 Monitor Flag Set)
Central Exchange Jump to MA (C170 Monitor Flag Clear)

- *013jk Central Exchange Jump*

This instruction shall be executed as described in the C170 Engineering Spec. with MEJ/CEJ always enabled and with the following extension:

A state switch option is selectable in C170 monitor mode and is controlled with the X0 sign bit in the C170 monitor exchange package. When this option is selected (C170 X0 sign bit set), the CPU shall switch from C170 state to C180 state monitor immediately after execution of the exchange from C170 monitor to C170 job. When the C170 X0 sign bit is clear there is no change in the operation of a 013 instruction. This option is not selectable for the exchange from C170 job mode to C170 monitor mode.

When execution of this state switch option completes, the C170 P points to the first instruction of the C170 job process exchanged into the CPU in C170 state.

The System Call Flag is set in the MCR upon completion of this state switch operation. A VMID of "1" and the System Call flag identify this interrupt as a state switch operation initiated by C170 state.

The process of state switching through execution of the extended 013 instruction shall not be interrupted by a C170 Exchange Request. This condition and others like SIT may be recorded in the MCR, in addition to System Call, but shall not interrupt the state switch operation. Likewise, UCR conditions such as PIT shall not interrupt the process of state switching.

7.4 STATE SWITCHING BETWEEN CYBER 180 & CYBER 170

A C170 State process or procedure may be initiated from C180 State in one of two ways:

a. The C180 Monitor may exchange to a C170 State process, and initialize that process by means of the contents of the C180 Exchange Package which defines that process.

b. A C180 Job process may call or return to a C170 State procedure. The procedure is initialized by means of the contents of the A and X Registers of the caller, or by the stack frame save area if returning.

The processor may switch from C170 State to C180 State either via a C180 Exchange (7.4.3) or via a Trap (7.4.5).

The Monitor Mode bit in the Status Summary (SS) register shall be toggled between Monitor Process State (MPS) and Job Process State (JPS) during each C180 Exchange operation, whether from C170 State to C180 State or vice versa.

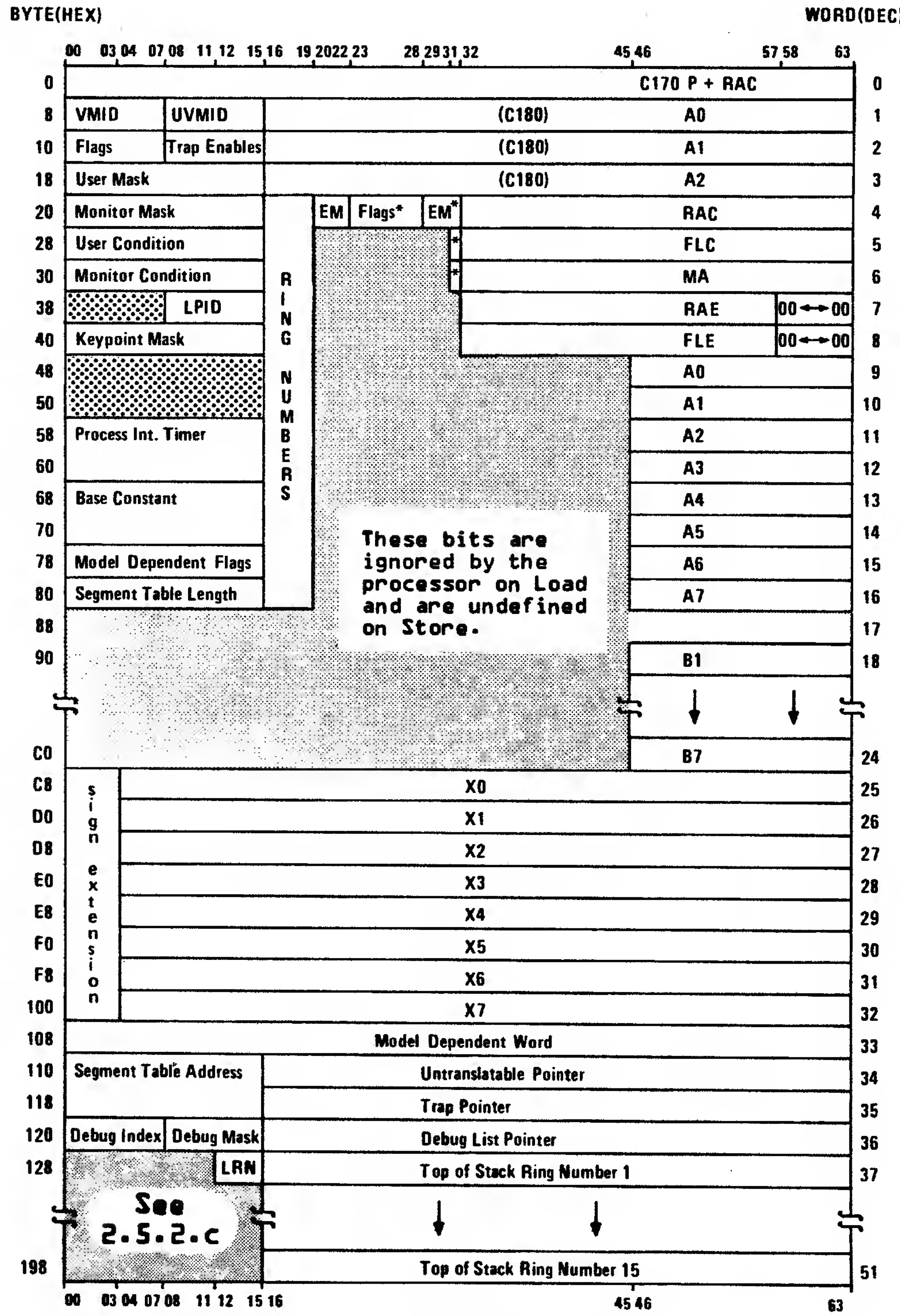
7.4.1 VMID

The process or procedure being initiated is defined to be in C170 State when the new VMID has a value of 1. (See 2.5.1 and 2.5.2.26.)

A VMCL equal to 1 indicates the processor's capability to run in C170 state but does not imply any information as to the PP actions relative to cache invalidation, interrupts, etc. The VMCL equal to 1 also implies no information as to whether the C170 environment is solo C170 or concurrent C170.

7.4.2 CYBER 180 Monitor to CYBER 170 State Exchange

The exchange shall be performed as defined in 2.6.1.6 and 2.8.5.2. Detection of the VMID value of 1 shall define the job process as a C170 State process. The C170 registers (A, B, X and miscellaneous) shall appear in the Exchange Package in the locations defined for C180 A and X Registers. See figure 7.4-1 for the format of this package. It will be noted that this is a C180 Exchange Package, because it exists within the C180 environment, and is the means by which the C180 Monitor communicates with the C170 State process. C180 bit numbering is used. The contents of the package are defined in the following paragraphs. Note that P, RAC, FLC, MA, RAE, and FLE are expressed in the number of words rather than in the number of bytes.



**For EM and Flags, see table 7.4-1.*

Figure 7.4-1. CYBER 170 State Exchange Package Mapping

Location in C180 Exch. Pkg. Word	Bit	Bit Name	Location in C180 Exch. Pkg. Word	Bit
4	20	EM-Parity Error (7.4.2.4)	3	59
	21	EM-Parity Error (7.4.2.4)		58
	22	EM-Parity Error (7.4.2.4)		57
	23	UEM Enable Flag		56
4	24	ESM Mode Flag	3	55
	25	Block Copy Flag		54
	26	Software Flag		53
	27	Inst. Stack Purge Flag		52
4	28	Software Flag	3	51
	29	EM-Indefinite Operand		50
	30	EM-Infinite Operand		49
	31	EM-Address Out of Range		48
5	31	C170 Monitor Flag	(not present)	
6	31	Exit Mode Halt	(not present)	

Table 7.4-1. Exchange Package Flags

7.4.2.1 P Register

The CYBER 180 P register shall be located in bits 00-63 of word 0 in the C180 Exchange Package. It shall contain a key, a ring number, and a segment number as defined in 2.1.1.1. The C180 P Register also contains the PVA within the CYBER 170 State process where execution is to begin. As such, the BN field of the C180 P register is not the C170 P address of a CYBER 170 job or monitor process. Instead bits 40-60 of C180 P contain the C170 P address of a CYBER 170 job or monitor process plus RAC of that process. Bits 32-39 and bit 63 shall contain zeros.

Bits 61 and 62 denote the C170 instruction parcel at which execution is to begin. C180 Exchange or Trap operations from C170 State to C180 State may occur between any two C170 instructions. When a C180 Exchange or Trap occurs, these bits shall be set as defined in sections 2.8.1 and 2.8.3.

An ECS instruction shall be interrupted between ECS Records (see 7.3.4 and 7.13) to provide faster interrupt response. When this occurs, the P stored shall point to the ECS instruction that was interrupted.

7.4.2.2 Stack Pointers

Registers A0 (C180), A1 (C180), and A2 (C180) shall be 48-bit CYBER 180 registers. They shall be located in bits 16-63 of words 1-3 respectively in the C180 Exchange Package. The contents of these registers are required to maintain stack integrity (see 7.4.4 and 7.4.5). In addition, they provide the capability to perform a Trap operation to a CYBER 180 procedure.

7.4.2.3 RAC, FLC, MA

C170 registers RAC, FLC and MA shall be located in bits 32 through 63 of C180 registers A3, A4 and A5 respectively. These quantities in the C180 Exchange Package for a C170 process represent word addresses or word counts (as opposed to byte numbers). The processor operation is undefined when:

$$0 > RAC \geq 10{,}000{,}000_8$$
OR
$$0 > FLC \geq 10{,}000{,}000_8$$
OR
$$0 > MA \geq 1{,}000{,}000_8.$$

7.4.2.4 Exit Mode

The Exit Mode (EM) field holds the C170 exit mode selections for a CYBER 170 State process. This field shall be located in bits 20-22 and 29-31 of word 4 of the CYBER 180 Exchange Package. The bits are defined as follows:

Bit 20:	*Parity Error (not used)*	*EM C170 Bit 59*
Bit 21:	*Parity Error (not used)*	*EM C170 Bit 58*
Bit 22:	*Parity Error (not used)*	*EM C170 Bit 57*
Bit 29:	*Indefinite operand*	*EM C170 Bit 50*
Bit 30:	*Infinite operand*	*EM C170 Bit 49*
Bit 31:	*Address out of range*	*EM C170 Bit 48*

Note that bits 20-22 refer to parity error conditions in CYBER 170. In the CYBER 170 State environment, these errors shall be handled by setting bit 48 of the Monitor Condition Register. (See section 7.6.) Bits 20-22 are loaded when entering the C170 environment and are copied through (stored unaltered) when leaving the C170 environment.

7.4.2.5 RAE, FLE

C170 registers RAE and FLE shall be located in bits 32 through 63 of C180 registers A6 and A7 respectively. These quantities in the C180 Exchange Package for a C170 process represent word addresses or word counts (as opposed to byte numbers).

In UEM (ECS mode) or in ECS, the processor operation is undefined for the execution of the 011, 012, 014, and 015 instructions when:

$0 > RAE \geq 10{,}000{,}000_8$ $(2{,}097{,}152_{10}$ *words)*
OR
$0 > FLE \geq 100{,}000{,}000_8$ $(16{,}777{,}216_{10}$ *words)*
OR
the lower six bits of RAE or FLE are nonzero.

Additional limits are placed on the sum of RAE plus FLE for UEM (ECS mode) as described in 7.2.4.2.3.

In UEM (ESM mode), the processor operation is undefined for the execution of the 011, 012, 014, and 015 instructions when:

$0 > RAE \geq 10{,}000{,}000{,}000_8$ $(1{,}073{,}741{,}824_{10}$ *words)*
OR
$0 > FLE \geq 4{,}000{,}000{,}000_8$ $(536{,}870{,}912_{10}$ *words)*
OR
the lower six bits of RAE or FLE are nonzero.

Additional limits are placed on the sum of RAE plus FLE for UEM (ESM mode) as described in 7.2.4.2.4.

7.4.2.6 A0-A7

Registers A0-A7 shall be 18-bit CYBER 170 registers. The quantities in the C180 Exchange Package for a C170 process represent word addresses (as opposed to byte numbers). They shall be located right-justified in bits 46-63 of words 9-16 in the CYBER 180 Exchange Package.

7.4.2.7 B1-B7

Registers B1-B7 shall be 18-bit CYBER 170 registers. They shall be located right-justified in bits 46-63 of words 18-24 in the Exchange Package. Register B0 is defined to be zero and does not exist in the CYBER 180 Exchange Package.

7.4.2.8 X0-X7

Registers X0-X7 shall be 60-bit CYBER 170 registers. They shall be located right-justified in bits 4-63 of words 25-32 in the Exchange Package. Bits 0-3 of each X Register shall contain the sign extension of bit 4. On a C180 Exchange from C170 State to C180 State, the processor shall store the C180 Exchange Package into memory with sign-extended X Registers. On a C180 Exchange from C180 State to C170 State, the processor may assume X Register sign extension in the C180 Exchange Package read from memory.

7.4.2.9 Control Flags

The following control flags shall be located in the exchange package:

Bit 23, Word 4: *UEM Enable Flag. In the one state, this flag shall enable the 011, 012, 014, and 015 instructions to access Unified Extended Memory rather than Extended Core Storage (see 7.2.3).*

Bit 24, Word 4: *ESM Mode Flag. In the one state, this flag shall select ESM mode rather than ECS mode in Unified Extended Memory (see 7.2.3). This flag being set in a processor having only the ECS mode shall cause the 011, 012, 014, and 015 instructions to be ILLEGAL (as shown in table 7.2-2 and figure 7.3-1 and 7.3-2) and shall not change the interpretation of RAE and FLE in the C170 Exchange Package (7.5.4).*

Bit 25, Word 4: *Block Copy Flag. In the one state, this flag indicates that C170 bits 30-50 of X0 shall be used instead of A0 to provide central memory addressing on 011 and 012 instructions. (See 7.3.4.)*

Bit 26, Word 4: *Software Flags. The specific definitions for these two reserved flags shall be contained*
Bit 28, Word 4 *in the software documentation. The two bits shall map respectively into C170 bits 53 and 51 of word 3 of the C170 Exchange Package, are loaded when entering the C170 environment and are copied through (stored unaltered) when leaving the C170 environment.*

Bit 27, Word 4: *CYBER 170 Instruction Stack Purge Flag. In the one state, this flag indicates that the additonal instruction stack purges described in paragraph 7.7 are to be performed.*

Bit 31, Word 5: *CYBER 170 Monitor Flag. In the one state, this flag shall indicate that the CYBER 170 State process is to begin (or resume) executing in CYBER 170 Monitor Mode. (Refer to the CYBER 170/173 Engineering Specification for the definition of the Monitor Flag.)*

Bit 31, Word 6: *Exit Mode Halt. In the one state, this flag indicates that a CYBER 170 State process was interrupted by a programming error condition which would have caused a CYBER 170 processor to halt. (See 7.6 for details.) The Exit Mode Halt flag is a message from a CYBER 170 State process to the C180 Monitor.*

The processor shall ignore the state of this flag. However, if this flag is set in the C180 exchange package for a process beginning execution, then the final state of this flag when the process ceases execution is undefined.

Note: *This bit will not be set for hardware error exits which would have caused a CYBER 170 processor to halt.*

7.4.2.10 CYBER 180 Ring Numbers

The CYBER 180 A Register ring number must not be altered in CYBER 170 State.

7.4.3 CYBER 170 State to CYBER 180 Monitor Exchange

An exchange from a CYBER 170 State process to CYBER 180 Monitor may be initiated by either of two conditions:

a. *Setting of an MCR and/or UCR bit as defined in tables 2.8.1 and 2.8.2.*

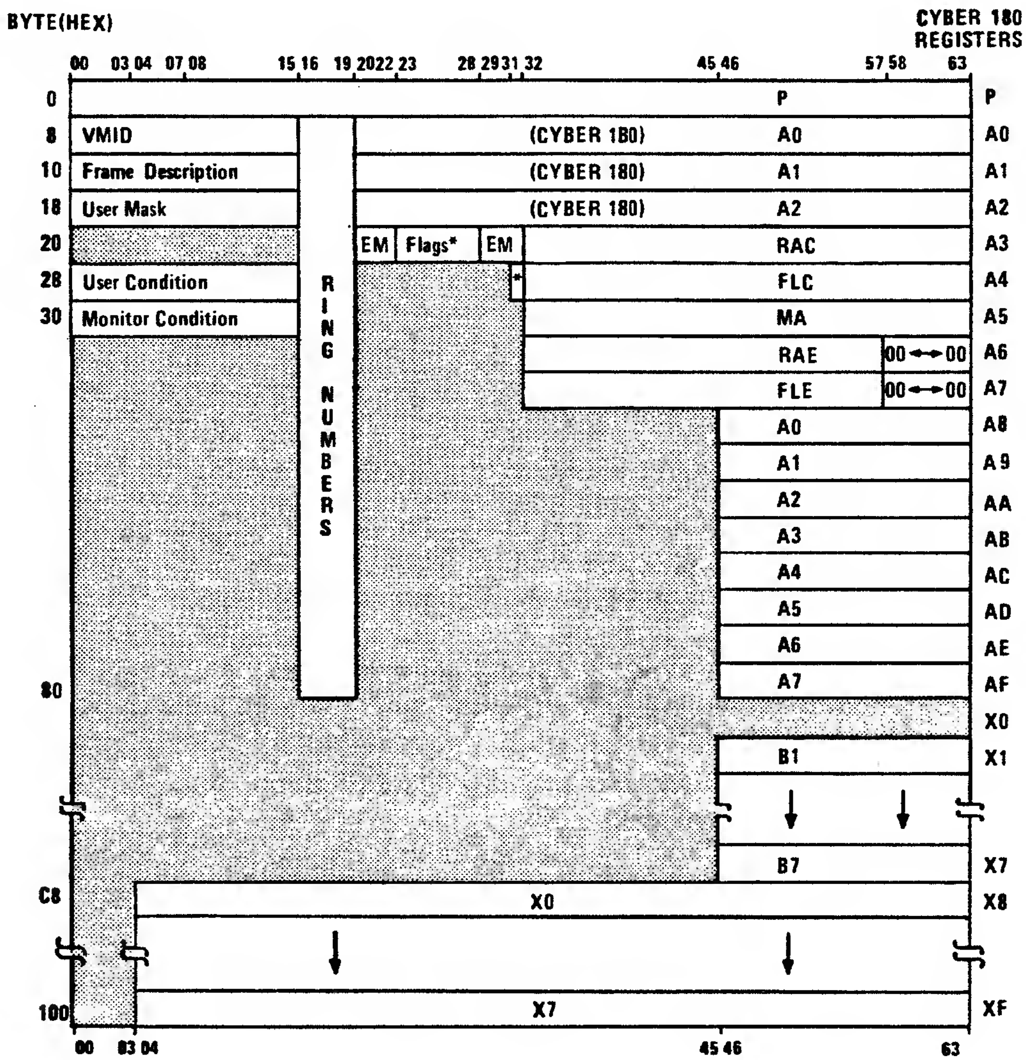
b. *Any programming error within the CYBER 170 State process which would cause a CYBER 170 central processor to halt. See 7.6 for a list of these errors.*

The exchange from the CYBER 170 State process to CYBER 180 Monitor shall proceed as described in 2.8.5.1. The format of the C180 Exchange Package which is stored in central memory shall be according to figure 7.4-1. Except for those situations described in paragraphs 7.2.5.2 and 7.2.5.3, the CYBER 170 State process may be reinitiated by the CYBER 180 Monitor at a later time by means of the procedure specified in 7.4.2. The contents of the CYBER 170 Exchange Packages within the CYBER 170 State process are not updated during the exchange from the CYBER 170 State process to CYBER 180 Monitor. (See 7.5.) Only the CYBER 180 Exchange Package which defines the CYBER 170 State process shall be modified.

7.4.4 Call to a CYBER 170 State Procedure from a CYBER 180 Job

The Call operation shall be performed as described in 2.6.1.2. Parameters shall be passed to the CYBER 170 procedure by means of the CYBER 180 A and X Registers. Figure 7.4-2 defines the format expected in these registers at the time of the Call operation. The leftmost four bits of the 64-bit X Registers within the processor are undefined at the beginning of the Call operation. Thus, they do not necessarily contain sign extension from the 60-bit X Registers. Register B0 is defined to be zero.

The BN field of the Code Base Pointer (which becomes the BN for the P of the called CYBER 170 process) is not the P address of the CYBER 170 process but, instead, bits 40-60 contain the P word address plus the RAC of that process. Bits 32-39 and 61-63 shall contain zeros.



*Notes: *For EM and Flags, see table 7.4-1.*
For RAE and FLE, see paragraph 7.5.4

Figure 7.4-2. C180 Stack Frame Save Area Containing C170 Environment

The format shown is for TRAP. There is no CALL from a C170 process. Undefined fields in the Stack Frame Save Area may be changed to undefined values during the execution of CALL instructions and shall be ignored during the execution of a Return instruction. This also includes bits 2 and 3 of the SFSA Descriptor.

7.4.5 Trap from CYBER 170 State to CYBER 180 State A

A Trap condition arising during a C170 State process shall result in a Trap operation as described in section 2.8.6. The C170 registers (A, B, X and miscellaneous) shall be stored into the Stack Frame Save Area in the locations normally defined for the C180 A and X Registers. The format for the C170 registers is shown in figure 7.4-2. The leftmost four bits of the 64-bit X Registers within both the Stack Frame Save Area and the processor are undefined at the conclusion of the Trap operation.

The BN of the P stored into the Stack Frame Save Area is not the P address of the interrupted CYBER 170 process but, instead, bits 40-63 contain the P address plus the RAC of that process. Bits 32-39 and bit 63 shall contain zeros. Bits 61 and 62 shall denote the CYBER 170 instruction parcel as defined in 2.8.

7.4.6 Return to a CYBER 170 Process

A Return operation from CYBER 180 to CYBER 170 shall be performed as specified in 2.6.1.4. The VMID from the Stack Frame Save Area having a value of 1 shall cause the process being reinitiated to be interpreted as a CYBER 170 process. The initiating segment must have global privileges to initiate a Return to CYBER 170. The CYBER 170 registers (A, B, X, and miscellaneous) shall be taken from the Save Area (or from the final values of the CYBER 180 A and X Registers after the Return operation) according to the format shown in figure 7.4-2. The leftmost four bits of the 64-bit X registers taken from the Stack Frame Save Area, or the final values of the C180 X registers after the Return operation are undefined.

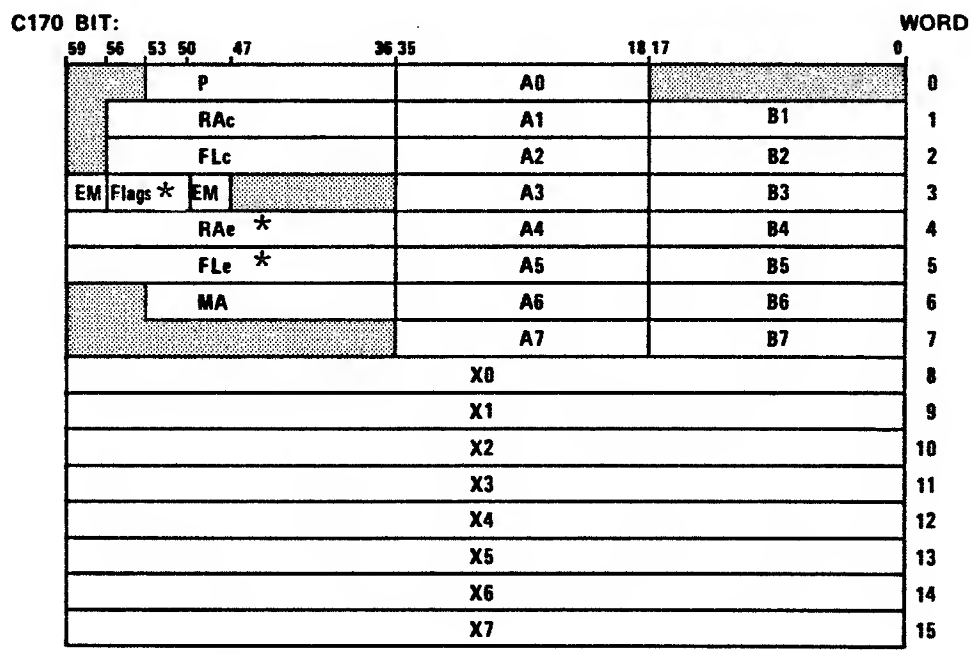
The BN of the PVA in P taken from the Stack Frame Save Area is not the P address of the CYBER 170 process but, instead, bits 40-63 contain the P address plus the RAC of that process. Bits 32-39 and bit 63 shall contain zeros. Bits 61 and 62 of the BN denote the CYBER 170 instruction parcel at which execution is to begin.

7.5 CYBER 170 EXCHANGE

7.5.1 CYBER 170 Exchange Packages

The CYBER 170 exchange packages shall exist within the CYBER 170 Memory Image (7.2). The format of the CYBER 170 Exchange Package shall conform to figure 7.5-1.

The contents of one CYBER 170 Exchange Package shall be in the processor registers during the execution of a CYBER 170 Mode process. Any C170 environment which is not being executed shall be represented by a C170 Exchange Package mapped into either a C180 Exchange Package or Stack Frame Save Area (see 7.4-1, 7.4-2).



**Notes: For EM and Flags, see table 7.4-1.*
For RAE and FLE, see paragraph 7.5.4

Figure 7.5-1. CYBER 170 Exchange Package

7.5.2 CYBER 170 Exchange Jump

The CYBER 170 Exchange Jump shall be initiated according to CDC 19063000, i.e. the exchange shall be initiated by:

a. CPU op code 013 when executing a procedure or process which has VMID equal to 1.

b. PPU op codes 2600, 2610, 2620 when the CPU is executing a procedure or process which has VMID equal to 1. (See 7.12. When a PP initiates a CYBER 170 exchange, the CPU shall complete executing instructions within the current instruction word that are not ECS or ESM data transfer instructions before performing that exchange.

c. Illegal instructions or software Exit Mode conditions within a CYBER 170 job process as defined in CDC 19063000.

The CYBER 170 Exchange Jump shall perform an Exchange Jump between two CYBER 170 programs.

The exchange shall proceed as follows:

a. The CYBER 170 registers (A, B, X, and miscellaneous) of the current job shall be packed into the 16-word format of figure 7.5-1. The leftmost four bits of all 64-bit words in this 16-word format are undefined.

b. This 16-word package shall be swapped with the 16-word package at the exchange address. I/O initiated references to central memory need not be blocked during the C170 Exchange operation.

c. The register values in the new package shall be unpacked and loaded into the processor's registers.

d. The C170 Monitor Flag shall be toggled, unless the exchange was initiated by PPU op code 260X, in which case it is unchanged.

The C180 Monitor Mode bit of the CYBER 180 Processor Status Summary register shall not be altered by this action (2.5.1.13). Similarly, the contents of the CYBER 180 Exchange Package (at JPS) associated with the CYBER 170 State process shall not be altered.

See 7.14.1 and 7.14.2 for extensions to the C170 exchange required to support concurrent C170 environments.

7.5.3 Undefined Fields

All fields which are indicated to be undefined in figure 7.5-1 are stored into memory as zeros and are ignored when loaded into the processor's registers. In addition, Register B0 is defined to be zero.

7.5.4 RAE, FLE

Processors without UEM (ESM mode), or processors with UEM (ESM mode) installed and the ESM Mode flag clear shall interpret RAE and FLE to and from the C170 Exchange Package as follows:

59	56 42	41 36
////	20 RAE 6	000000

59 42	41 36
23 FLE 6	000000

Processors with UEM (ESM mode) installed and the ESM Mode flag set shall interpret RAE and FLE to and from the C170 Exchange Package as follows:

59   36
29   RAE   6

59   36
29   FLE   6

Note that RAE and FLE for UEM (ESM mode) are right-shifted six bit positions from their representations when not using UEM (ESM mode).

7.6 ERROR HANDLING IN CYBER 170 STATE

Error handling shall comply with CDC 19063000 except as detailed below.

7.6.1 Program Errors which Cause CPU Halt

Within a CYBER 170 State process, program errors which are defined in CDC 19063000 to result in a CPU halt shall, instead, cause the processor to store the Exit Mode bits and the P register contents in location RAC and then to perform a C180 Exchange to C180 Monitor. These errors are:

a. *Illegal instruction with Monitor Flag set.*

b. *Read or Write Address Out of Range with Monitor Flag set and Exit Mode selected for this error.*

c. *RNI or Branch Address Out of Range with Monitor Flag set.*

d. *Infinite or Indefinite with Monitor Flag set and Exit Mode selected for this error.*

e. *00 instruction with Monitor Flag set.*

The Exit Mode Halt Flag shall be set in the CYBER 180 Exchange Package associated with the CYBER 170 State process. The contents of P stored into the C180 Exchange Package shall point to the instruction as specified in table 7.6-1.

7.6.2 Hardware Errors

Hardware errors which set bits 48 or 50 of the Monitor Condition Register shall cause the interrupt specified in table 2.8-1. (These errors shall not store either the Exit Mode bits or the P Register contents into location RAC of a currently executing CYBER 170 State process.)

7.6.3 Error Exit - C173/C170 State of C180

See table 7.6-1, sheets 1, 2, and 3.

Error Conditions	Mode	C173 Error Response: Exit Mode Selected	C173 Error Response: Exit Mode Not Selected	C170 State of C180 Error Response: Exit Mode Selected	C170 State of C180 Error Response: Exit Mode Not Selected
Illegal instructions. See Note 1.	Monitor	1. Execute illegal instruction as a pass. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point either to the word containing the illegal instruction or to the following word. 4. Clear P. 5. Halt processor.		1. The illegal instruction is not executed. 2. Stop instruction execution. 3. Store C170P and exit condition bits at location RAC. C170P at location RAC will point to the word containing the illegal instruction. 4. Perform C180 Exchange with C180P = RAC + address of illegal instruction.	
	Job	1. Execute illegal instruction as a pass. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point either to the word containing the illegal instruction or to the following word. 4. Clear P. 5. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 6. Resume instruction execution.		Identical to C173 except that C170P at location RAC will always point to the word containing the illegal instruction.	
Exit condition C170 bit 48 set by an increment read with an address out of range (AOR). Op Code 5X.	Monitor	1. Read all zeros to selected X register. 2. The A register contains the AOR address. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. P will point either to the word containing the increment instruction or to the following word. 5. Clear P. 6. Halt processor.	1. Read all zeros to selected X register. 2. Continue execution. See Note 2.	1. The X register is unchanged. 2. The A register contains the AOR address. 3. Stop instruction execution. 4. Store C170P and exit condition bits at location RAC. Identical to step 4 of the C173. However, note that due to the two possibilities for C170P, C170P will not necessarily be identical in all instances. 5. Perform C180 Exchange with C180P = RAC + address of incr. instruction OR RAC + address of instruction following the increment.	Identical to C173 except that the selected X register is unchanged. See Note 2.
	Job	1. Read all zeros to selected X register. 2. The A register contains the AOR address. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. P will point either to the word containing the increment instruction or to the following word. 5. Clear P. 6. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 7. Resume instruction execution.		Identical to C173 except that the selected X register is unchanged. However, note that due to the two possibilities for C170P, C170P at location RAC will not necessarily be identical in all instances.	

Table 7.6-1 Error Exits, C170 Monitor & Job Modes (1 of 7)

Error Conditions	Mode	C173 Error Response: Exit Mode Selected	C173 Error Response: Exit Mode Not Selected	C170 State of C180 Error Response: Exit Mode Selected	C170 State of C180 Error Response: Exit Mode Not Selected
Exit Condition C170 bit 48 set by a Direct Read Central Memory instruction with an address out of range (AOR). Op Code 660.	Monitor	Instruction 660 is executed as a pass.		1. The Xj register is unchanged. 2. The Xk register contains the AOR address. 3. Stop instruction execution. 4. Store C170P and exit conditions bits at location RAC. C170P will point either to the word containing the 660 instruction or to the following word. 5. Perform a C180 Exchange with C180P = RAC + address of 660 instruction OR RAC + address of instruction following the 660 instruction which had the AOR.	1. The Xj register is unchanged. 2. Continue execution.
	Job			1. The Xj register is unchanged. 2. The Xk register contains the AOR address. 3. Stop instruction execution. 4. Store C170P and exit condition bits at location RAC. C170P will point either to the word containing the 660 instruction or to the following word. 5. Clear 170P. 6. Exchange jump to MA and set MF; C170P stored into the C170 Exchange package equals zero. 7. Resume instruction execution.	
Exit condition C170 bit 48 set on CMU instruction (models 172 through 174 only). 1. C1 or C2 greater than 9. 2. R1 or R2 address out of range.	Monitor	1. Error condition 1 causes the instruction to execute as a pass. Error condition 2 causes instruction moves or compares up to the point of address out of range. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point either to the CMU instruction or to the word following the CMU instruction. 4. Clear P. 5. Halt processor.	1. Error condition 1 causes the instruction to execute as a pass. Error condition 2 causes instruction moves or compares up to the point of address out of range. 2. Continue with the next instruction. See note 2.	1. Error condition 1 causes the instruction to execute as a pass. Error condition 2 may cause partial instruction execution (moves or compares) no further than up to the point of address out of range. P will point either to the CMU instruction or to the word following the CMU instruction. 2. Stop instruction execution. 3. Store C170P and exit condition bits at location RAC. Identical to step 3 of the C173. However, note that due to the two possibilities for C170P, the C170P will not necessarily be identical to the C173 in all instances. 4. Perform C180 Exchange with C180P = RAC + address of CMU instruction OR address of the word following the CMU inst'n.	1. Error condition 1 causes the instruction to execute as a pass. Error condition 2 may cause partial instruction execution (moves or compares) no further than up to the point of address out of range. 2. Continue with the next instruction. See note 2.
	Job	1. Error condition 1 causes the instruction to execute as a pass. Error condition 2 causes instruction moves or compares up to the point of instruction out of range. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point either to the CMU instruction or to the word following the CMU instruction. 4. Clear P. 5. Exchange jump to MA and set MF: P stored into the C170 Exchange Package equals zero. 6. Resume instruction execution.		Identical to C173. However, note that due to the two possibilities for C170P, the C170P at location RAC will not necessarily be identical to the C173 in all instances. The partial transfer or compare on error condition 2 will also not necessarily be identical to the C173.	

Table 7.6-1 Error Exits, C170 Monitor & Job Modes (2 of 7)

Error Conditions	Mode	C173 Error Response: Exit Mode Selected	C173 Error Response: Exit Mode Not Selected	C170 State of C180 Error Response: Exit Mode Selected	C170 State of C180 Error Response: Exit Mode Not Selected
Exit condition C170 bit 48 set by an ECS address range check for instructions 011 and 012.	Monitor	1. Execute ECS instruction as a pass. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point to the word following the ECS instruction. 4. Clear P. 5. Halt processor.	1. Execute ECS instruction as a pass. 2. Exit to next 60-bit word and continue execution. See Note 2.	1. No data shall be transferred. 2. Store C170P and exit condition bits at location RAC. C170P at location RAC will point to the word containing the 011/012 or to the following word. 3. Perform C180 Exchange with C180P = RAC + address of word containing the 011/012 instruction OR RAC + the address of the following word.	Identical to C173. See Note 2.
	Job	1. Execute ECS instruction as a pass. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point to the word following the ECS instruction. 4. Clear P. 5. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 6. Resume instruction execution.		Identical to C173 except that C170P stored at RAC will point to the word containing the 011/012 or to the following word rather than always to the word following the 011/012.	
Exit condition C170 bit 48 set by an ECS address range check for instructions 014 and 015.	Monitor	Instructions 014 and 015 are executed as illegal instructions.		1. No data shall be transferred. 2. Store C170P and exit condition bits at location RAC. C170P at location RAC will point to the word containing the 014/015 or to the following word. 3. Perform C180 Exchange with C180P = RAC + address of the 014/015 instruction OR RAC + address of the following instruction.	1. Execute 014/015 as a pass. 2. Exit to next parcel and continue execution. See Note 2.
	Job			1. No data shall be transferred. 2. Store C170P and exit condition bits at location RAC. C170P at location RAC will point to the word containing the 014/015 instruction on to the following word. 3. Clear C170P. 4. Exchange jump to MA and set MF; C170P stored into the C170 Exchange Package equals zero. 5. Resume instruction execution.	

Table 7.6-1 Error Exits, C170 Monitor & Job Modes (3 of 7)

Error Conditions	Mode	C173 Error Response: Exit Mode Selected	C173 Error Response: Exit Mode Not Selected	C170 State of C180 Error Response: Exit Mode Selected	C170 State of C180 Error Response: Exit Mode Not Selected
Exit condition C170 bit 48 set by an increment write with an address out of range (AOR). Op Code 5X.	Monitor	1. Block write operation; content of CM is unchanged. 2. The A register contains the AOR address. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. P will point either to the word containing the increment instruction or to the following word. 5. Clear P. 6. Halt processor.	1. Block write operation; content of CM is unchanged. 2. The A register contains the AOR address. 3. Continue execution. See Note 2.	1. Block write operation; content of CM is unchanged. 2. The A register contains the AOR address. 3. Stop instruction execution. 4. Store C170P and exit condition bits at location RAC. Identical to step 4 of the C173. However, due to the two possibilities for C170P, C170P will not necessarily be identical in all instances. 5. Perform C180 Exchange with C180P= RAC + address of increment instruction OR RAC + address of instruction following the increment.	Identical to C173. See Note 2.
	Job	1. Block write operation; content of CM is unchanged. 2. The A register contains the AOR address. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. P will point either to the word containing the increment instruction or to the following word. 5. Clear P. 6. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 7. Resume instruction execution.		Identical to C173 except that the selected X register is unchanged. Note that due to the two possibilities for C170P, C170P at location RAC will not necessarily be identical in all instances.	
Exit Condition C170 bit 48 set by a Direct Write Central Memory instruction with an address out of range (AOR). Op Code 670.	Monitor	Instruction 670 is executed as a pass.		1. Block write operation; content of CM is unchanged. 2. The Xk register contains the AOR address. 3. Stop instruction execution 4. Store C170P and exit condition bits at RAC. C170P will point either to the word containing the 670 instruction or to the following word. 5. Perform a C180 Exchange with C180P = RAC + address of 670 instruction OR RAC + address of instruction following the 670 instruction which had the AOR.	1. Block write operation, content of CM is unchanged. 2. The Xk register contains the AOR address. 3. Continue execution.
	Job			1. Block write operation; content of CM is unchanged. 2. The Xk register contains the AOR address. 3. Stop instruction execution. 4. Store C170P and exit condition bits at location RAC. C170P will point either to the word containing the 670 instruction or to the following word. 5. Clear C170P. 6. Exchange jump to MA and set MF; C170P stored into the C170 Exchange package equals zero. 7. Resume instruction execution.	

Table 7.6-1 Error Exits, C170 Monitor & Job Modes (4 of 7)

<table>
<tr><th rowspan="3">Error Conditions</th><th rowspan="3">Mode</th><th colspan="2">C173</th><th colspan="2">C170 State of C180</th></tr>
<tr><th colspan="2">Error Response</th><th colspan="2">Error Response</th></tr>
<tr><th>Exit Mode Selected</th><th>Exit Mode Not Selected</th><th>Exit Mode Selected</th><th>Exit Mode Not Selected</th></tr>
<tr><td rowspan="2">Exit condition C170 bit 48 set by an RNI.</td><td>Monitor</td><td colspan="2">1. Stop instruction execution.
2. Store P and exit condition bits at location RAC. P will point to the word required by the RNI.
3. Clear P.
4. Halt Processor.</td><td colspan="2">1. Stop instruction execution.
2. Store C170P and exit condition bits at location RAC. C170P at location RAC will point to the word required by the RNI.
3. Perform C180 Exchange with C180P= RAC + address of the instruction required by the RNI.</td></tr>
<tr><td>Job</td><td colspan="2">1. Stop instruction execution.
2. Store P and exit condition bits at location RAC. P will point to the word required by the RNI.
3. Clear P.
4. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero.
5. Resume instruction execution.</td><td colspan="2">Identical to C173.</td></tr>
<tr><td rowspan="2">Exit condition C170 bit 48 set by a branch address out of range.</td><td>Monitor</td><td colspan="2">1. Stop instruction execution.
2. Store P and exit condition bits at location RAC. P will point either to the word containing the branch instruction or to the following word. See note 3.
3. Clear P.
4. Halt processor.</td><td colspan="2">1. Stop instruction execution.
2. Store C170 and exit condition bits at location RAC. C170P at location RAC will point either to the word containing the branch instruction or to the word at the branch destination. (Note that for the Return Jump either K or K+1 is acceptable.) See note 3.
3. Perform C180 exchange with C180P= RAC plus address of the branch instruction OR RAC plus address of the branch destination. (Note that for the Return Jump either K or K+1 is acceptable).</td></tr>
<tr><td>Job</td><td colspan="2">1. Stop instruction execution.
2. Store P and exit condition bits at location RAC. P will point either to the word containing the branch instruction or to the following word. See note 3.
3. Clear P.
4. Exchange jump to MA and set MF. P stored into the C170 Exchange Package equal zero.
5. Resume instruction execution.</td><td colspan="2">Identical to C173 except for the C170P stored at RAC which points either to the word containing the branch instruction or to the word at the branch destination. (Note that for the Return Jump either K or K+1 is acceptable.)</td></tr>
</table>

Table 7.6-1 Error Exits, C170 Monitor & Job Modes (5 of 7)

Error Conditions	Mode	C173 Error Response: Exit Mode Selected	C173 Error Response: Exit Mode Not Selected	C170 State of C180 Error Response: Exit Mode Selected	C170 State of C180 Error Response: Exit Mode Not Selected
Breakpoint signal from CMC	Monitor	1. Execute remaining parcels of 60-bit word currently executing. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. 4. Clear P. 5. Halt processor.		No hardware breakpoint is available.	
	Job	1. Execute remaining parcels of 60-bit word currently executing. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. 4. Clear P. 5. Exchange jump to MA and set MF. P stored into the C170 Exchange Package equals zero. 6. Resume instruction execution.			
Infinite condition, C170 bit 49. Indefinite condition, C170 bit 50.	Monitor	1. Execute instruction. 2. Update Xi. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. P will point either to the word containing the arithmetic instruction or to the following word. 5. Clear P. 6. Halt processor.	1. Execute instruction. 2. Update Xi. 3. Continue execution. See Note 2.	1. Execute instruction. 2. Update Xi. 3. Identical to step 4 of the C173. Note that due to the two possibilities for C170P, C170P will not necessarily be identical in all instances. 4. Perform C180 Exchange with C180 = RAC + address of arithmetic instruction OR RAC + address of instruction following the arithmetic instruction.	Identical to C173 See Note 2.
	Job	1. Execute instruction. 2. Update Xi. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. P will point either to the word containing the arithmetic instruction or to the following word. 5. Clear P. 6. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 7. Resume instruction exerution.		Identical to C173. However, note that due to the two possibilities for C170P, C170P at location RAC will not necessarily be identical in all instances.	
ECS flag register parity, C170 bit 51. CMC to CPU data parity error or double error, C170 bit 53.	Monitor	1. Execute instruction. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point either to the word containing the instruction or to the following word. 4. Clear P. 5. Halt processor.	1. Execute instruction. 2. Continue execution.	1. Set Detected Uncorrectable Error, MCR bit 48. See Table 2.8-1. The processor shall not set C170 bits 51/53.	
	Job	1. Execute instruction. 2. Stop instruction execution. 3. Store P and exit condition bits at location RAC. P will point either to the word containing the instruction or to the following word. 4. Clear P. 5. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 6. Resume instruction execution.			

Table 7.6-1 Error Exits, C170 Monitor & Job Modes (6 of 7)

		C173		C170 State of C180	
		Error Response		Error Response	
Error Conditions	Mode	Exit Mode Selected	Exit Mode Not Selected	Exit Mode Selected	Exit Mode Not Selected
CPU to CMC address or data parity error, or CPU to CM address parity error, C170 bit 52.	Monitor	1. Block write operation; content of CM is unchanged. 2. Block read operation forces read data to all ones. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. 5. Clear P. 6. Halt processor.	1. Block write operation; content of CM is unchanged. 2. Block read operation forces read data to all ones. 3. Continue execution. See Note 2.	1. Set Detected Uncorrectable Error, MCR bit 48. See Table 2.8-1. The processor shall not set C170 bit 52.	
	Job	1. Block write operation; content of CM is unchanged. 2. Block read operation forces read data to all ones. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. 5. Clear P. 6. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 7. Resume instruction execution.			
CPU to CMC address parity error on exchange jump address, C170 bit 52.	Monitor	1. Write operation is not blocked. 2. Block read operation of first word forces read data to all ones. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. 5. Clear P. 6. Halt Processor.	1. Write operation is not blocked. 2. Block read operation of first word forces and read data to all ones. 3. Rest of exchange jump executes normally. See Note 2.	1. Set Detected Uncorrectable Error, MCR bit 48. See Table 2.8-1. The processor shall not set C170 bit 52.	
	Job	1. Write operation is not blocked. 2. Block read operation of first word forces read data to all ones. 3. Stop instruction execution. 4. Store P and exit condition bits at location RAC. 5. Clear P. 6. Exchange jump to MA and set MF; P stored into the C170 Exchange Package equals zero. 7. Resume instruction execution.			
00 instruction	Monitor	1. Stop processor execution. 2. Store P and exit condition bits at location RAC. P will point either to the word containing the 00 instruction or to the following word. 3. Clear P. 4. Halt processor.		1. The 00 instruction is not executed. 2. Store C170P and exit condition bits at location RAC. C170P at location RAC will point to the word containing the 00 instruction 3. Perform C180 Exchange with C180P-RAC + address of 00 instruction.	
	Job	1. Stop processor execution. 2. Store P and exit condition bits at location RAC. P will point either to the word containing the 00 instruction or to the following word. 3. Clear P. 4. Exchange jump to MA and set MF. P stored into the C170 Exchange Package equals zero. 5. Resume instruction execution.		Identical to C173 except that C170P stored at RAC always points to the word containing the 00 instruction.	

Table 7.6-1 Error Exits, C170 Monitor & Job Modes (7 of 7)

Notes for Error Exit Charts

1. *The ILLEGAL instructions in the C170 State of C180 are:*

   - *011-013 at parcel 1, 2, or 3*
   - *011, 012 for certain error conditions described in figure 7.3-1*
   - *014, 015 for certain error conditions described in figure 7.3-2*
   - *Any 30-bit instruction in parcel 3*

The ILLEGAL instructions in C173 are:

- *014, 015, 016, 017*
- *011-013 at parcel 1, 2, or 3*
- *011, 012 and no ECS present*
- *Any 30-bit instruction in parcel 3*

(Note that the C173 does "execute" these instructions; see C170 Reference Manual.)

2. *When an Exit Condition occurs and the corresponding Exit Mode bit is not set and therefore execution continues, the Exit Condition bit may, but need not be retained until either the next:*

   Error Exit at which time the bit for which Exit Mode was not selected will be stored at RAC along with the bit which caused the Error Exit.

   Exchange (C170 or C180) or Trap operation at which time the bit for which Exit Mode was not selected will be discarded.

This is a difference between C173 and the C170 State of 180 because in C170 the Exit Condition bit would remain set, at least to the end of an instruction word. Not only is there no requirement to retain the Exit Condition bit that is not selected, but also note that in the C170 State of C180, a C180 interrupt may occur between any two C170 instructions and the Exit bit, even if retained, will be discarded because of the interrupt.

3. *When an exchange occurs to an environment with FLC = 0, the subsequent instruction fetch will result in Address Out of Range. The required store into RAC will then take place ignoring this FLC restriction.*

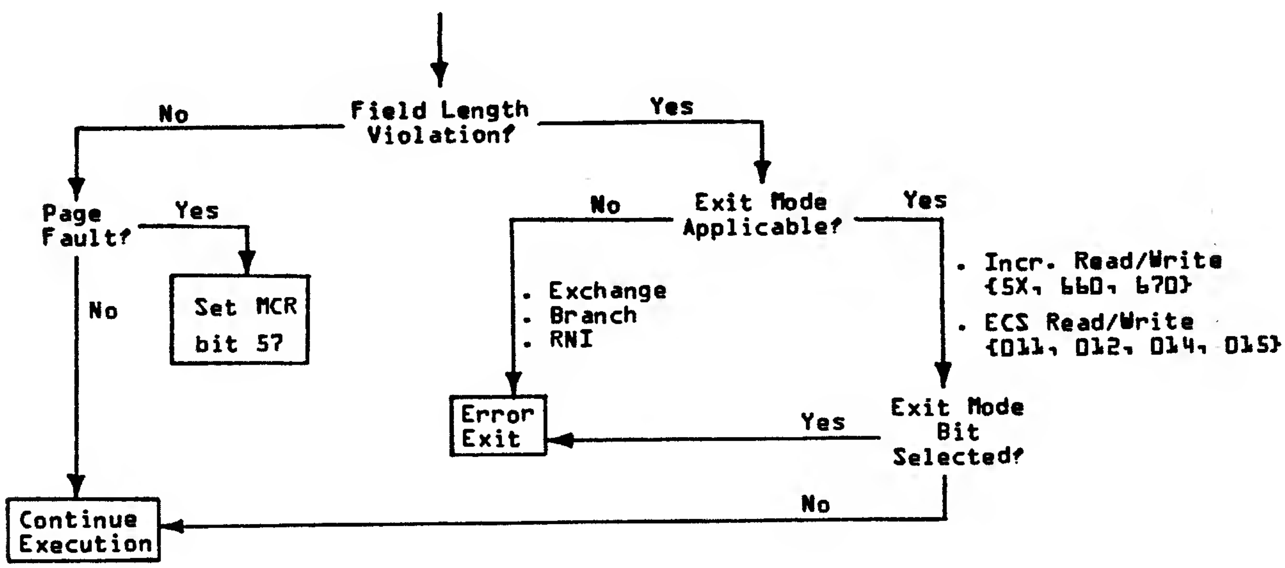
7.6.4 Address Out of Range

An Address Out of Range condition shall be detected and appropriate action taken as described in 7.6.3 whenever

C170 Address ≥ FLC

Refer to figures 7.3-1 and 7.3-2 for the handling of Address Out of Range conditions in ECS/ESM.

An Address Out of Range because of FLC or FLE violations when the memory references would also cause a Page Table Search without Find shall Error Exit or Pass as specified for the C170 rather than react to the Page Fault by setting MCR bit 57 (except for the 014 instruction). For the 014 instruction, the processor may report either Page Fault or Address Out of Range when both conditions are present.

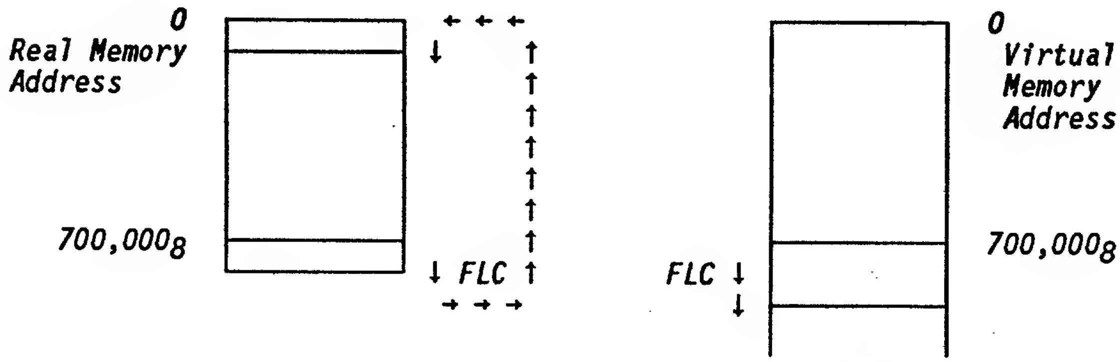


The C173 address arithmetic is 18-bit ones complement. In the C170 State of C180, the address arithmetic for incrementing P+RAC and for adding an address to RAC has been extended to 21 bits to support up to 2M word Central Memory portion of the C170 Memory Image. The following points should be noted.

RAC + FLC > $777,777_8$

In the C173, a combination of RAC and FLC for which RAC + FLC > $777,777_8$ will allow central memory references to wrap around as shown in the example below. These same combinations in the C170 State of C180, as shown below, will not wrap around.

C173 | C170 State



C173		C170 State	
FLC =	200 000	FLC =	0 200 000
RAC =	700 000	RAC =	0 700 000
Reference Address =	100 000	Reference Address =	0 100 000
Real Memory Address =	$000\ 001_8$	Virtual Address =	$1\ 000\ 000_8$

C170 P > $777{,}777_8$

In the C173, P is never greater than $777{,}777_8$ because C170 P is 18 bits and also because C170 P must be less than FLC which is 18 bits. The extension of FLC to 21 bits in the C170 State of C180 allows P to increment to values greater than $777{,}777_8$. If P is greater than $777{,}777_8$ and a C170 Exchange occurs, the P stored into the C170 Exchange Package is truncated to 18 bits.

RAC = 200,000
FLC = 2,000,000

	P relative to RAC "C170-like"	P+RAC "C180-like"
P incrementing ↓	777,776	1,177,776
↓	1,000,000	1,200,000
↓	1,000,001	1,200,001
↓	1,000,002	1,200,002

A C170 Exchange during this period cannot store the actual C170 P into the C170 Exchange Package because it now exceeds 18 bits.

Last Word of 262K Memory

In the C173, the last word, $777{,}777_8$ of central memory cannot be accessed from the processor. This same word is accessible in C170 State of C180 as follows:

INSTRUCTION FETCH

C173		C170 STATE	
FLC =	200 000	FLC =	0 200 000
RAC =	700 000	RAC =	0 700 000
P =	77 776	P =	0 077 776
P+RAC =	777 776	P+RAC =	0 777 776
P+1+RAC =	000 000	P+1+RAC =	0 777 777
P+2+RAC =	000 000	P+2+RAC =	1 000 000

OPERAND ADDRESS ARITHMETIC

C173		C170 STATE	
FLC =	100 000	FLC =	0 100 000
RAC =	700 000	RAC =	0 700 000
Operand Address =	077 777	Operand Address =	077 777
Address + RAC =	000 000	Address + RAC =	0 777 777

Last Word of the Central Memory Portion of the C170 Memory Image Segment

The RAC and FLC restrictions in paragraph 7.2.3 exclude the last word (address 7,777,777$_8$) in the 2M word C170 Memory Image and all addresses above that word from the usable C170 Memory Space. In the case of RAE and FLE, the last 64$_{10}$ words are excluded when in UEM (ECS mode).

```
                                  0 000 000
                                      :
                                      :
                                  7 777 677₈  Highest Address Available via
                                              RAE/FLE in UEM (ECS mode)
                                  7 777 700₈
                                      :
                                      :
                                  7 777 776₈  Highest Address Accessible via
                              ____             RAC/FLC
                             |    7 777 777₈  Last word of 2M word C170 Memory
          Available only     |        :       Image
          via RAE/FLE in     |        :
          UEM (ESM mode)     |        :
                             |11 000 377 675₈ Highest Address Accessible via
                             |____             RAE/FLE in UEM (ESM mode)

                                   RAE 7 777 777 700
                                    X0 1 777 777 777
                Block Size -1                377 776
                                      ______________
                                      11 000 377 675₈
```

7.6.5 Parcel Boundaries

Tests will be performed on 30-bit instructions to ensure that they do not begin in the last parcel of a word and hence, cross word boundaries. Tests will be performed on 60-bit instructions and the 011, 012 instructions to ensure that they begin in parcel 0. CYBER 170 State instructions which cross word boundaries are defined as illegal.

7.7 CODE MODIFICATION IN CYBER 170 STATE

Any model-dependent instruction stacks shall always be purged by execution of a CYBER 170 Return Jump, ECS Read, Exchange Jump, or Long Jump instruction (Op. codes 010, 011, 013, 02).

If the Instruction Stack Purge flag (7.4.2.9) is set, the instruction stack and the instruction pipeline shall also be purged immediately following the execution of the instructions listed below:

1. *Any conditional jump instruction (Op. codes 03 through 07) for both branch and fall-through conditions.*
2. *Any CPU store instruction (Op. codes 50-57, i=6,7).*

Notes: *1. For the case of a store instruction in parcel 0, 1, or 2 which modifies its own location in central memory (the address of the store is equal to the current value of P), the execution is different from the C170 processors. The C170 state of C180 will always execute the modified instruction word following the completion of the store instruction when the Instruction Stack Purge Flag is set and may execute the modified word if a C180 interrupt occurs even when the Purge Flag is not set. (The C170 processor will always execute the unmodified remaining parcels in the instruction word.)*

2. The 670 instruction never causes an instruction stack purge.

7.8 CEJ/MEJ

All CYBER 170 instructions which are defined to be conditional on the state of the CEJ/MEJ switch shall assume that this switch is permanently enabled.

7.9 DEBUG

The CYBER 180 Debug feature shall not be supported in CYBER 170 State.

7.10 CYBER 170 BREAKPOINT

The CYBER 170 breakpoint features shall not be supported.

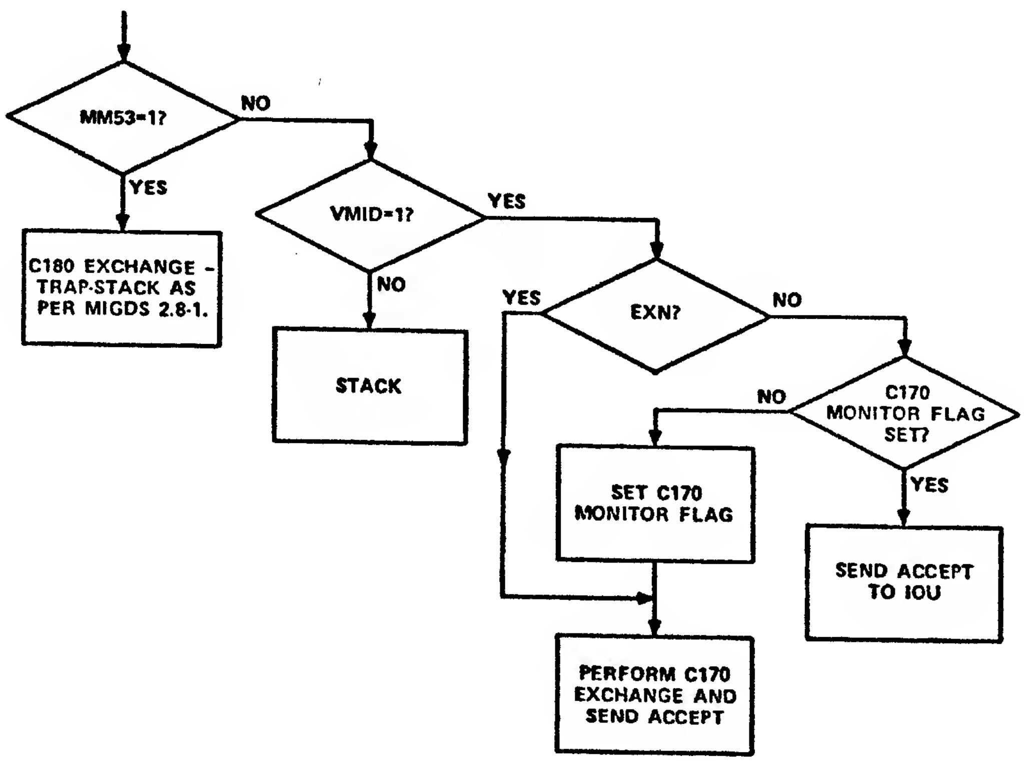
7.11 READ CYBER 170 P REGISTER

This feature is not supported by the hardware. Hence, it is not possible for the IOU to directly read the CYBER 170 relative P address (CYBER 170 PPU instruction 27x). Note that the IOU may read the CPU P address (a PVA) by using the Maintenance Channel.

7.12 CYBER 170 PP EXCHANGE REQUESTS

There are three PP exchange instructions defined for CYBER 170 Mode: Exchange Jump (260X), Monitor Exchange Jump (261X), and Monitor Exchange Jump to MA (262X). The IOU shall initiate each instruction by setting its Exchange Busy bit and sending an Exchange Request signal to the CPU. The Exchange Busy bit, which is cleared by an Exchange Accept signal from the CPU, prevents overlapping of Exchange Request processing within the CPU. The three exchange instructions are further described in paragraphs 7.12.1 to 7.12.3 and 5.8.2. The specific processor to which these interrupts are directed is specified in 7.14.3.

The C170 Exchange Request signal from the IOU shall set bit 53 of the Monitor Condition Register and then initiate the following action.



The PP actions relative to the central memory and the processor shall be kept in order for an individual PP. Thus, a write operation followed by an interrupt from the same PP must be executed in the order issued by the PP.

- *When the processor is in C170 Monitor Mode with MM 53 clear and the C170 Exchange Request is a MAN or MXN, no C170 Exchange operation is required. The processor shall return the Exchange Accept to the IOU with no significant processor performance degradation. (A small time penalty, less than the time required for a C170 Exchange operation incurred once during each contiguous time period spent in C170 Monitor would not be considered significant. Any time penalty incurred upon each MXN or MAN would be considered significant.)*
- *When a C170 Exchange operation results from the C170 Exchange Request:*
  1. *The C170 Exchange operation shall occur only between C170 instruction words or between data blocks for ECS instructions (see paragraph 7.13.4).*
  2. *The Exchange Accept shall be sent to the IOU at the completion of the C170 Exchange.*
- *The hardware will examine the C170 Exchange Request before execution of any C170 instructions when returning to C170, calling into C170 or executing a C180 Exchange to C170 with the starting P address on a C170 word boundary. When executing a C180 Exchange to C170 with the starting P address not on a C170 instruction word boundary, any required C170 Exchange will not occur until the beginning of the next C170 instruction word.*
- *C180 Exception conditions shall result in an Exchange or Trap between any two C170 instructions or shall cause the termination of an ECS instruction (which would then be restarted from the beginning upon return to the interrupted C170 procedure, see 7.13.4). The C180 Exchange or Trap shall have priority over the C170 Exchange Request if present.*

7.12.1 Exchange Jump

For an Exchange Jump instruction (2600), the IOU shall send an Exchange Request signal, a 00 Exchange Code, and an 18-bit Exchange Address (word address) to the CPU. This address, which is the starting location of the CYBER 170 Exchange operation, shall be left-shifted three places by the CPU and used as the BN portion of an address in the segment used for CYBER 170 State execution. The state of the Monitor Flag is unaffected by this exchange. (See 7.5.2.) An Exchange Accept signal is returned to the IOU at the completion of the C170 Exchange.

7.12.2 Monitor Exchange Jump

For a Monitor Exchange Jump instruction (2610), the IOU shall send an Exchange Request signal, a 01 Exchange Code, and an 18-bit Exchange Address (word address) to the CPU. If the Monitor Flag is clear, the exchange is performed as described in paragraphs 7.12.1 and 7.5.2, with the exception that the Monitor Flag is then set. If the Monitor Flag is already set, the exchange is not performed. An Exchange Accept signal is returned to the IOU at the completion of the C170 Exchange or immediately when already in C170 Monitor Mode.

7.12.3 Monitor Exchange Jump to MA

For a Monitor Exchange Jump instruction (2620), the IOU shall send an Exchange Request signal, a 10 Exchange Code, but no 18-bit address to the CPU. The MA register shall be used as an 18-bit word address which is the starting location of the CYBER 170 Exchange operation. If the Monitor Flag is clear, the exchange is performed as described in paragraph 7.5.2, and the Monitor Flag is then set. If the Monitor Flag is already set, the exchange is not performed. An Exchange Accept signal is returned to the IOU at the completion of the C170 Exchange or immediately when already in C170 Monitor Mode.

7.13 EXTENDED CORE STORAGE (ECS) COUPLER

Transfers of data between ECS and the central memory shall be initiated by the CPU and driven by the ECS coupler. The coupler shall be a physical equipment located remote from the CPU and central memory, and shall be connected to both by coaxial cables. When ECS is present, execution of ECS instructions, as seen by standard software, shall be as specified in 7.3.

A detailed description of the CYBER 180 ECS Coupler and of the CYBER 180 CPU/Coupler interface is contained in the CYBER 180 ECS Coupler Equipment Specification, CDC 11897699.

7.13.1 Interface to Central Memory

The ECS coupler shall interface to central memory by the standard memory port defined in section 4.1.3 of this specification. External Interrupts appearing on this port shall be ignored. The CYBER 180 ECS Coupler Interface Requirements Specification, CDC 11896624, covers the handling by the coupler of error response codes received from the central memory port.

7.13.2 Interface to CPU

The interface to the CPU shall consist of 12 coaxial lines. The interface shall use the synchronous AC transmission scheme defined in CDC 52318500 (CYBER 180 Processor/Memory Transmission Scheme Specification). Transmissions shall be synchronized to the clock of the memory port to which the coupler is connected (and hence, to the clock of the CPU).

The signals of this interface are defined as follows:

a. *Signals from CPU to Coupler*

Control Byte	*8 bits (+ 1 Parity)*
Request	*1 bit*

b. *Signals from Coupler to CPU*

Full Exit	*1 bit*
Half Exit	*1 bit*
Error End of Operation	*1 bit*
Corrected Error	*1 bit*

7.13.3 Initiating or Terminating from CPU

The CPU shall initiate or terminate transfers by sending a block of 8 Control Bytes to the coupler. The 8 Control Bytes shall contain the values of: the starting address in central memory, the starting address in ECS (which includes flag operation information), the length of the transfer, and a Write ECS bit. The control information is packed as shown in figure 7.13-1; byte number 0 is transmitted first.

The Write ECS bit, when set, shall indicate that the ECS transfer is to proceed from central memory to ECS. When clear, the transfer shall proceed from ECS to central memory.

The processor shall divide transfers greater than 64 words in length into a series of shorter blocks (none of which shall exceed 64 words in length) so as to provide greater interrupt response. For each of these blocks, the processor shall transmit the control bytes (figure 7.13-1) to the ECS Coupler and then wait for the coupler to appropriately respond when finished. Before initiating the transfer of another block, the processor shall then test for any outstanding interrupt, and if present, interrupt the ECS instruction. When an ECS instruction is interrupted and subsequently restarted, it must be executed completely from the beginning.

The processor shall:

1. *Continue initiating data blocks until Bj+K=0 on ECS READ whenever HALF or FULL EXIT is received at the end of each data block.*
2. *Stop initiating data blocks whenever an ERROR END OF OPERATION is received at the end of a data block on either ECS READ or WRITE.*
3. *Stop initiating data blocks whenever a HALF EXIT is received at the end of a data block on ECS WRITE.*
4. *Continue initiating data blocks until Bj+K=0 on ECS WRITE whenever FULL EXIT is received at the end of each data block.*

The processor shall set up the word count for the data blocks such that any interrupt that occurs will be taken only between ECS Records but never between the last two ECS Records.

The CPU shall do all of the RA and FL range checking for ECS instructions.

When the ECS starting address for an ECS Read has C170 bit 21 set, then the processor shall transfer zeros directly to central memory without involving the ECS coupler. When the ECS starting address for an ECS Read is less than 2^{21} and the transfer terminates in nonexistent memory (including not physically installed), the ECS Coupler/Controller shall execute the zero transfer as required.

Write Bit (1)
↓

Data	Length (8)	Starting Central Memory Addresses (23)			/////// ///////	Starting ECS Address (24)		
Byte No.	0	1	2	3	4	5	6	7

Figure 7.13-1. ECS Protocol

7.13.4 Cache Purge

After initiating an ECS transfer which copies data into central memory from physical ECS, the CPU shall purge the associated addresses in its Cache buffer.

7.13.5 Maintenance Channel and Registers

The ECS Coupler shall have its own maintenance registers and maintenance channel interface.

7.14 DUAL PROCESSOR C170 OPERATION

Some dual processor systems shall allow the second processor to execute in C170 state. Some of these systems will also allow the two processors to execute C170 concurrently. The term concurrent C170 is used to refer to this concurrent operation of two CPUs - both in C170 state. Solo C170 is used to refer to the C170 state which is not required to run in concurrent mode. All systems which support concurrent C170 will also support solo C170 in either processor. See section 1.6 for a list of systems supporting C170 state operations, and those supporting concurrent C170.

Processors which support solo C170 shall have one bit in the Options Installed register to indicate the presence of solo C170. Processors which support concurrent C170 shall have an additional bit to indicate the presence of the concurrent C170 capability.

While the specific bit numbers in the OI register may be model-dependent, the bits shall have the following definition:

Solo C170

Set - *This processor is available to execute solo C170.*

Clear - *This processor is not available to execute solo C170 state.*

Concurrent C170

Set - *This processor is available to execute C170 simultaneously with a concurrent C170 state in the other processor. A processor having this bit set may be operated by itself in solo C170. (See section 1.6.)*

Clear - *The processor is not available to execute concurrent C170. Processors not having this concurrent C170 Flag are not available to execute concurrent C170. (See section 1.6.)*

7.14.1 Cache Purge on C170 Exchange

Processors operating in concurrent C170 mode shall purge all C170 entries in the cache on each C170 exchange operation. This includes C170 job to C170 monitor and C170 monitor to C170 job. This does not include C180 exchanges to or from C180 monitor. The C170 entries are those entries with an ASID of $FFFF_{16}$ *as per 7.2.5.4.*

7.14.2 Concurrent C170 Monitor Interlock Flag

An interlock to prevent both C170 processors from running in C170 monitor mode simultaneously shall be present on processors in concurrent C170 operation.

This shall be implemented by redefining the global privilege bit in the C170 Segment Descriptor Entry as the C170 Monitor Interlock Flag. (Toggling this bit simply switches the C170 state execution between global privileged execution and unprivileged execution - a concept which has no meaning to C170.) The processors shall set this Interlock flag with a Test and Set Lock sequence on a C170 job-to-monitor exchange. A processor desiring to enter C170 monitor and finding the flag set shall remain in a loop testing the flag until the interlock flag clears. (See note below.) Being in a loop testing the Monitor Interlock Flag shall not prevent the processor from responding to a bit being set in the MCR (as per table 2.8-1) or to a maintenance channel function. This processor shall not set the Halt bit in the processor SS register while in the loop.*

There shall only be one Process Segment Table and, thus, only one Segment Descriptor Entry for the C170 environment. Both processors must be referencing the same entry to utilize the same flag. The processors must test the flag in memory and not in any MAP for this test for the same reason. The interlock flag shall be cleared when the processor leaves C170 monitor. This step is a simple clear, not a Test and Clear Flag sequence.

The hardware shall have no requirement to test this interlock flag on a C180 exchange into the C170 environment. (If the C180 to C170 exchange enters C170 job, the interlock flag will be tested on any subsequent C170 exchange to C170 monitor. If the C180 to C170 exchange enters C170 monitor and this processor was not exchanged previously out of monitor, thus, leaving the interlock flag set, the software, not the hardware, is responsible for interlocking.)

A processor in solo C170 mode may, but need not perform the above operation with the C170 Monitor Interlock Flag.

**Note - This information is incomplete and will be the subject of a future DAP. Contact AD&C for the current status of this DAP.*

7.14.3 PP Generated Interrupts

Systems having only solo C170 in the first processor:

2600, 2610, 2620 shall always be directed to the first processor.

2601, 2611, 2621 are undefined and may be directed to either or neither processor. It is a software responsibility to not issue these three instructions.

Systems having solo C170 in the first or second processor or having concurrent C170:

2600, 2610, 2620 shall always be directed to the first processor.

2601, 2611, 2621 shall always be directed to the second processor.

7.14.4 PP Initiated Cache Invalidations

Systems having only solo C170 in the first processor:

The cache invalidations as described in 7.2.5.4 need only to be implemented in the first processor.

Systems having solo C170 in the first or second processor or having concurrent C170:

The cache invalidations shall be implemented in both processors. (Note that this requires one cache invalidation stream from one IOU to two processors. This does not require the ability to accept a second cache invalidation stream from a second IOU.)

It may be necessary on a model-dependent basis to provide bits in the Memory Environment Control Register to disable the C170 cache invalidation. This will allow running diagnostics on a "downed" processor without the continual stream of cache invalidations. This would also be useful, if not necessary, for performance reasons to allow disabling the C170 cache invalidations to a processor dedicated to C180 tasks.

8.0 RELIABILITY, AVAILABILITY, MAINTAINABILITY (RAM)

The scope of RAM is to reduce failure rates of hardware and software, to increase the availability of the system to the customer, and to achieve the lowest system maintenance costs while improving first fix effectiveness. Fault-tolerant and fault-avoidance techniques which yield the greatest increase in RAM per cost, and which are within the guidelines of total expenditures for RAM features should be used.

8.1 DEFINITIONS

8.1.1 System States

8.1.1.1 Fully Operational

A fully operational system is capable of rated throughput with no faults present.

8.1.1.2 Fault-tolerant Operation

A fault-tolerant system is capable of automatic recovery from, and operating with, faults present while having no discernable impact on throughput. Throughput impact is restricted to the recovery process time.

8.1.1.3 Degraded Operation

A degraded system is able to continue correct operation by reducing function or lowering throughput. Fault tolerance is preferred over degradation.

8.1.1.4 Down

A down system is unable to maintain data integrity or operational correctness as a result of a fault occurrence. It is assumed that all environmental conditions for proper operation are met.

8.1.2 Interrupt

An interrupt is a fault occurrence that causes a system to cease to perform and requires an unscheduled manual intervention to resume operation. Manual intervention included operator interaction with the system error recovery process and unscheduled maintenance actions. It is assumed that all externally provided environmental conditions for proper operation are met.

8.1.3 Deferred Maintenance

Deferred maintenance is scheduled maintenance specifically intended to eliminate an existing fault which did not prevent continued successful operation of the system. If the deferred maintenance process can not be accomplished concurrently with the user tasks, the time to repair is considered down time, but the maintenance event is not considered to be an interrupt.

8.2 MINIMUM RAM FEATURES

8.2.1 Maintenance Processor

The presence of another processor referred to as a Maintenance Processor on all systems is required to ensure that the maintenance personnel or program can interrogate the system in the case of a failure of the central processors.

8.2.2 SECDED

Single error correction/double error detection (SECDED) shall be implemented on main memory. Errors shall be reported to the Maintenance Processor when error correcting and whether the error is a single or double occurrence. (See section 9.0 for reporting requirements.)

8.2.3 Parity Checking

Parity shall be checked on all data paths, address paths, channels, and registers.

8.2.4 Degradable Cache and Map

Cache buffer and Map buffer shall have the capability of having portions of them degraded.

The CPU shall also have the capability of bypassing Cache or Map or both (degraded operation).

8.2.5 Fault Isolation

Error signals which localize faults shall be provided for O.S. and Maintenance Processor so that appropriate degradation or reconfiguration may take place and maintenance action time can be minimized. Error isolation to a module level shall be possible for 95 percent of all solid failures by use of hardware and software to localize a single fault. Error detection circuitry shall be designed so that errors do not propagate beyond the next interface in the system before they are detected, which will minimize the hardware checks required to localize the fault.

8.2.6 Reconfiguration and Degradation

When permanent failures occur, the system shall be reconfigured by a combination of hardware and software techniques (goal is to be fully automatic).

All functional components (adders, busses, etc.) shall be designed with reconfiguration and degradation ideas in mind in case of failure. If an arithmetic operation such as a divide were to fail, reconfiguration in the form of subtraction could possibly be used to emulate the divide.

Reconfiguration due to failing hardware shall include, to the fullest extent practical, suppression or elimination of any error indication from the failing hardware which has been reconfigured out of use.

Reconfiguration is a way of maintaining availability by avoiding a down state. Reconfiguration/ Emulation within the CPU is only possible if the error detection logic can localize the fault so that ambiguity does not exist as to what is to be reconfigured. This becomes more difficult when failures occur outside a well defined unit such as an adder with error detection, etc.

8.2.7 Instruction Retry

A combination of hardware and software techniques shall be used to retry failing instructions. At a minimum, the hardware shall detect failing instructions and provide an error signal to the Maintenance Processor for software implemented retry of instructions and logging of error type that has occurred. Error signals shall cause an interrupt to an error handling routine (software). Software will attempt a retry if hardware reports an uncorrected error (in Status Summary or MCR48=DUE) and the process is not damaged (PND). Refer to section 2.5.2.5 for details. This shall occur during the failing instruction so as not to allow following instructions to alter registers or memory. The address of instruction which caused the error interrupt shall be included in the exchange package.

An error status register shall be implemented with access by the Maintenance Processor which will indicate what type of error occurred, such as instruction failure, memory read error, single or double error, etc. The Central Processor shall provide accurate first failure data capture in the error status registers.

Any automatic hardware retry circuitry shall provide a selective enable/disable which is controlled by the Maintenance Processor. The default condition for automatic hardware retry is the disabled state. Software or Maintenance Processor initiated retry is the preferred error recovery approach for NOS/VE systems. (See 9.1.2.2.)

8.2.8 Micro Step Mode

A Maintenance Processor controlled Micro Step Mode shall be implemented so as to allow micro program control instruction execution starting at any microcode address. Any number of micro instructions can then be executed, including single micro instructions.

The Error Correction Code (ECC) shall also be checked on each address contents.

8.2.9 Time-out

Whenever one system facility is connected to another via command/response protocol, a time-out mechanism shall be provided to ensure continuing operation of the system.

8.2.10 Power Supplies

All cabinets shall have individual power supplies, and circuit breakers shall be used in place of fuses. This philosophy will extent to all fused circuits in the machine.

The Maintenance Processor shall have command and control capability for remote power-on and power-off of each mainframe element.

8.2.11 Packaging

The number of module types shall be held to a minimum so as to reduce spares cost, increasing the likelihood of available module types on hand in the event of failure. All like modules shall be fully interchangeable and replaceable when power is on.

While an individual mainframe element may be down, the system shall continue to operate for all tolerated faults. Power and packaging design shall facilitate concurrent repair of these faults without impacting the execution of user tasks.

Easy access to all field replaceable units (FRU) shall be provided.

Circuit board differences within the module (if more than one circuit card per module) shall also be held to a minimum to simplify manufacturing. Chip layout on the circuit boards shall be standardized for the purpose of reducing hole patterns to be drilled, therefore, reducing artwork layout costs and manufacturing costs.

8.2.12 Forced Errors

NOTE: The following section is not a requirement for S0, S1, S2, S3, THETA and PIP3. This section is intended for all subsequent C180 systems. Previous products may or may not comply.

There shall be a method of forcing fault conditions into the fault-tolerant hardware added to the system, so that the operation of all fault handling mechanisms can be verified in-house for Product/Design Verifications and in the field for confidence testing and validation of repair actions. The mainframe fault injection requirements are:

- Fault injection shall be accomplished under software control, during the execution of the operating system.
- Injected faults reported by a CPU shall appear to have the following characteristics:
  - With Process Not Damaged (PND set)
    a) Transient error "correctable" by hardware retry (i.e. the error condition is created by a one-shot.)
    b) Transient error "correctable" by software retry (i.e. the error condition is present until hardware retry has been exhausted but then drops at this point.)
    c) Solid error (i.e. the error condition is still present after software retry has been exhausted.)
  - With Process Damaged (PND clear)
    a) Transient error "correctable" by software retry (i.e. the error condition drops as software is notified)
    b) Solid error (i.e. the error condition is still present when software has exhausted its alternatives.)
- Error status bits reported in each mainframe element's Status Summary shall be able to be forced on an individual element basis. Each individual type of error condition detectable and reported in the Detail Error Status shall be excerised. Errors shall be injected at the lowest level and their impact should be confined whenever possible. Examples are: Memory induced error in a single word, error in the next occurrence of an instruction, or error the next time a functional unit is used. Although the injection of errors may be random, the error condition shall manifest itself as a single occurrence.
- The Detected Uncorrected Error (DUE) status reported in the Monitor Condition Register (MCR) shall be able to be forced under the following conditions:
  a) Executing in C180 Monitor Mode with traps enabled.
  b) Executing in C180 Job Mode with traps enabled.
  c) Both conditions "a and b" with traps disabled.
  d) Mainframes with dual state capability shall allow the selection of forcing an error occurrence in either C170 State (both C170 Monitor and C170 Job Modes) or C180 State when executing in C180 Job Mode.

- Memories with SECDED shall support insertion of the following fault conditions which are denoted by the syndrome code. This applies to all memories in the mainframe and is not restricted to main/central memory.

  a) Valid single bit error
  b) Double bit error
  c) Invalid single bit error (multiple bit failure).

- Injected faults do not need to be conditioned to occur at a specific program address, on a specific instruction in a code sequence or at a fixed time/cycle interval. Conditioning shall be on a functional basis. Functional conditioning means that the next time that function is executed a fault will be injected. If different fault handling algorithms are the result of multiple system conditions occuring at the same time (system state/mode), the hardware shall provide for the combinational conditioning. Specific conditions are model-dependent and will occur as a result of unique hardware and software architectures. These conditions shall be documented in the appropriate hardware specifications.

- Injection of multiple faults in a single system element at a given time is not supported. (Exception is double or multiple bit errors in memory.) Multiple system faults may exist in different system elements at the same time.

  The basic assumption for the CYBER 180 RAM design philosophy is the only one fault exists per error handling mechanism at a time.

- Environmental problems such as temperature and power shall be simulated.

- Since stress testing exceeds environmental limits of protection circuitry, the design of the sensing circuits shall include a hardware disable function for each environmental control to facilitate in-house testing.

8.2.13 Programmable Clock Margins

Clock frequency must be ±2 percent program adjustable (via Maintenance Access).

For the purpose of Design Verification, the clock frequency must be ±3 percent program adjustable.

8.2.14 Component Failure Rates

Component failure rates shall be obtained from the latest revision to MIL-HDBK-217 and in accordance with CDC Standard 1.12.006.

8.2.15 Remote Technical Assistance (RTA)

The mainframe shall be designed to support RTA having the following requirements:

- a single telephone line access to the mainframe with a minimum data transfer rate of 2400 bps
- the system shall support concurrent RTA and system operation
- RTA shall have remote deadstart capability
- RTA shall provide remote access to error data, system dump analysis, downed element diagnostic analysis while operating system is on-line, and access to peripheral equipment
- auto dial and security features shall be site selectable

8.2.16 Hardware Redundancy

Static redundancy is provided by two or more copies of hardware performing the same function at the same time. Either through additional detectors or output voting techniques (employed where three or more copies exist). The correct output is automatically gated. Errors detected within this configuration are reported as corrected errors.

Dynamic redundancy is implemented using reconfiguration techniques and shall be controlled by a maintenance processor after a Stop on Error interrupt has occurred. Utilization of dynamic redundancy should not be dependent on the involvement of a failed element. An error detected in the active function is reported as an uncorrected error.

Back-up or "shadow" registers/memories are a form of redundancy used to ensure error containment. Like dynamic redundancy, errors detected in either set of data should be reported as an uncorrected error. Recovery will be attempted, where possible, by a maintenance processor after sufficient error information is collected during the Stop on Error period.

Selection of a redundant hardware technique is a model-dependent decision that is based on performance, reliability, and life-cycle cost trade-offs.

8.3 ENVIRONMENTAL FAILURES

CYBER 180 mainframe components (central processors, central memories and the IOU) shall monitor local environmental conditions such as local power, temperature, etc. The 50Hz/60Hz power source to the MG set shall also be monitored.

Environmental faults detected shall be divided into three classes:

- Faults which give no prior warning
- Faults which give short warning (2.5 seconds)
- Faults which give long warning (greater than 120 seconds)

For a description of short and long warning conditions see section 9.0.

Detailed error data defining the specific cause of the environmental fault condition detected shall be available to the Maintenance Processor. If total loss of power occurs, first failure environmental data shall be retained until the Maintenance Processor is able to capture this information.

8.4 PERFORMANCE MONITORING

Processors not having a Performance Monitoring Facility (PMF) shall make provision to measure performance on a model-independent basis. (See paragraph 1.6.) Examples of the items are cache hit rates, instruction execution rates, time in C180 monitor mode, and memory activity rates. The measurements may be made in-house and by external instrumentation rather than built-in hardware facilities, and are intended to provide feedback into the development process.

9.0 DEDICATED FAULT TOLERANCE (DFT)

The DFT/0S buffer is the operating system's primary means of obtaining information on the mainframe hardware error conditions. The buffer exists in main memory and is jointly controlled by a Maintenance Processor (PP or service processor) and the central processor(s). The format for this buffer and the protocol used for intercommunication is described in the DFT/OS Interface Specification.

Included in the DFT definition are the mainframe elements consisting of: Central Memory, IOU, Processors, Attached Processors, Shared/Global Memories. In addition, any external power and cooling support required by these elements shall also report into the DFT Maintenance Processor. Error handling for peripheral equipment connected to the I/O channels is the responsibility of the PP I/O driver software.

The purpose of this section is the description of the error reporting requirements for the mainframe hardware. The intent is to define the model-independent error reporting requirements for the interface between the mainframe hardware and the Maintenance Processor. These requirements are defined in two categories: element status summary and detailed error reporting.

9.1 ELEMENT STATUS SUMMARY

Each element should provide a single source of information that provides the global status for that element. This information should be available to the maintenance processor in one access to the specified element.

9.1.1 System Error Summary Reporting

This section specifies the error reporting strategy at a system level.

9.1.1.1 Long Warning

The objective of Long Warning is to protect the system integrity against the occurrence of all environmental faults. Long Warning is the reporting of an early warning to a system shutdown condition that would allow the operating system to achieve a recoverable state that will tolerate the complete shutdown of any or all of the system elements. Upon resumption of normal operating conditions, the operating system shall be able to recover operation of all active jobs in the system. The operating system relies on Central Memory for information pertaining to the subsystem and job environment status.

A Long Warning is necessary because the current Central Memory technology is a volatile memory. (Volatile means that information stored in the memory is lost if power to that memory is interrupted.) The Long Warning signal provides the operating system the opportunity to move the critical Central Memory information to nonvolatile storage.

In today's system architecture that means moving the data to disk. In order to be effective, the Long Warning indication shall be supported by each of the system critical elements. (Example: Processor, Main Memory, IOU, Mass Storage Controller/Device, and the Power/Cooling elements supporting these elements.) These elements are defined to be system critical because they must be operational in order to accomplish the defined task. The operating system designs require the following lengths of time to complete this task: NOS = 120 seconds, and NOS/VE = 120 seconds. (See equation below for more detailed description.)

The amount of time that the system must stay powered-on after the initial Long Warning indication is dependent on the operating system type, the type and number of mass storage devices and channels, and the size of Central Memory. Times stated in the previous paragraph represent the typical ballpark times. To determine the specific time for a known configuration follow these guidelines:

Determine the amount of time (t) that it would take to write out the entire contents of Central Memory to one of the configured disks. Then the total time required for Long Warning (T) is calculated as follows:

(T) = [t÷(2x)] + 30 seconds. Where x= number of mass storage channels available.

Assumptions:

1. NOS/VE Stand-alone Operating System.
2. Maximum of 50 percent of Central Memory resident pages will need to be written to disk.
3. Overhead will amount to 30 seconds.

Long Warning results from the environmental faults monitored for any conditions that may result in a system critical power shutdown and includes the following conditions, as applicable:

- High Temperature Warning
- Low Temperature Fault
- Blower Fault
- Condensing Unit Fault
- Power Failure

The reason for the Long Warning condition shall also be reported along with the Long Warning status.

9.1.1.2 Short Warning

Short and Long Warning provide for the same basic objective, the protection of system integrity against the occurence of all environmental faults. The distinction between the two conditions is the time duration from report of the warning condition to loss of power. For Short Warning this timeframe is 2.5 seconds.

In order to protect the file system, the operating system uses this time to achieve a defined state of inactivity. There is not enough time to save the contents of central memory on nonvolatile storage. If the environmental fault is corrected without a power shutdown of Central Memory, the system shall be able to restart without loss of integrity. If Central Memory integrity is lost, active job recovery is not possible.

All elements critical to the system's normal operation shall issue short warning. (Example: Processor, Main Memory, IOU, Mass Storage Controller/Device and the power/cooling support for these elements.) The processor may also take action on short warning via the MCR bit 50.

Short Warning results from the environmental faults monitored for any conditions that may result in a system critical power shutdown and include the following conditons, as applicable:

- High Temperature Warning
- Low Temperature Fault
- Blower Fault
- Condensing Unit Fault
- Power Failure

When only 2.5 seconds remains in the Long Warning period until power shutdown, a Short Warning indication should be given. The reason for the Short warning condition shall also be reported along with the Short Warning status.

9.1.1.3 Uncorrected Error

In the system error handling hierarchy, each element attempts to correct as many of the detected errors as possible. Reporting an uncorrected error means that the hardware was unable to correct the error. This signal provides the operating system the opportunity to attempt to correct the detected condition or proceed to a orderly halt state.

Uncorrected errors occur when the hardware has exhausted its correction resources. These include retry, correction codes, redundancy, degrade, and reconfiguration. Caution must be excercised by hardware in using degrade and reconfiguration to areas that are transparent to software operation, such as cache degrade, buffer and stack degrade or disable. If software's allocation of resources can be affected by degrade or reconfiguration, then the process of degrade or reconfiguration must be handled under software control. If hardware automatically reconfigures, degrades, or uses a redundant path, the reported corrective error status shall include notification of these actions so appropriate repair actions can be initiated.

Uncorrectable errors for the CPU error recovery process identified in section 9.1.2.2 are those detected errors that have not been inherently corrected by the error detection process. Only error correcting codes and static redundancy provide this capability. As a result, uncorrected errors will be reported when other hardware correction techniques (such as retry, reconfiguration, degrade) have not been employed. In order to fully utilize these other techniques, an error analysis step needs to be performed. If the failing CPU participates in its own error analysis, the potential for erroneous results is high. Error information gathered after the CPU has made an unsuccessful attempt may be undistinguishable. While the CPU reports an uncorrected error, it is the Maintenance Processor's responsibility to exercise every possible corrective action before reporting the error as uncorrected to system logs.

If software is expected to perform some sort of error correction utility, more information beside the Uncorrected Error status shall be required to help isolate the error condition. The Uncorrected Error signal serves the purpose of initiating the process. An uncorrected error shall be reported whether or not the element has judged the process to be damaged.

Improved containment of the error condition within an element shall be provided in order to achieve the best possible "first error capture" (stop on error, stop clocks, scan-out, OCMS). Processors stopped on error condition must be restartable by software without deadstart initialization.

The S0 processor provides a "first error" register. This register indicates the error which first caused instruction level retry (if any). By running with retry disabled, the Operating System can get information about the code and the environment at the time of the error. Software level retry can be accomplished with the assistance of DFT.

9.1.1.4 Corrected Error

The Corrected Error signal indicates that an error was detected in that element and the hardware was able to correct the error. Along with the Corrected Error status signal, the hardware is responsible to report sufficient information to define the type of error detected and isolate the fault condition.

The operating system takes no other action than to log the appropriate information reported by the hardware.

Concurrent hardware detection/correction techniques provide sufficient analysis information to accurately describe the failure and isolate that failure to a FRU. Additional environmental error data is not normally required. Software selectable stop on corrected error capability may be provided where more detailed error data may be of benefit.

9.1.2 Element Error Status Reporting

In the case of uncorrected error reporting, hardware has established that it was unable to handle the detected error condition. By reporting the general Uncorrected Error status, hardware is soliciting the aid of the next level of the system in tolerating the error. Alternative actions taken by the software may be different for the various hardware architectures in the CYBER 180 product line. However, there are some generic system state variables that need to be defined at the occurrence of the error condition that will lead the software analysis to the appropriate corrective action.

The following sections, specify these minimum hardware reporting requirements for each of the major elements in the mainframe. In addition to these minimum reporting requirements, the hardware shall also provide the model-dependent information required for unique CYBER 180 features.

9.1.2.1 Process Not Damaged (PND)

The processor element shall report Process Not Damaged status as defined in section 2.5.2.5.

9.1.2.2 Halt

The Processor and IOU elements shall report all processor halt conditions.

For either the central processor element or the IOU, a halt may result from an internal element condition (Table 2.8-1), an external Stop Processor Execution request (Section 6.2.1), or a stop on error condition. Fault tolerant operation requires that execution be restarted/resumed while maintaining the integrity of the operation. Exceptions are the PPs that can only be started by deadstart. If the PP memory is suspect, then that PP memory must be reloaded before deadstarting.

Future system designs (Theta-E1 and Vanguard) shall implement the CPU error recovery mechanism outlined in Figure 9.1-1. CPU Stop on Error is the key ingredient in this process. Figure 9.1-1 serves as a "road map" to more detailed MIGDS requirements. Section numbers for each reference are included in parentheses.

Stop on Error allows error analysis and error data collection to be independent of the operation of a failing CPU. Emphasis in each model-dependent CPU design should be placed on improved error containment. The objective is that every error can be retried/restarted without affecting other elements of the system or the integrity of the task.

Stop on Error is similar to a Halt. The Stop on Error is immediate and need not occur on an instruction boundary. When restarted, the current process performs a retry as prescribed by the hardware.

When the process cannot be retried (see Process Not Damaged, section 2.5.2.5.d), the maintenance processor will use the PND and DUE (MCR 48) as flags. The maintenance processor may force a monitor process to be restarted in the processor.

Stop on Error will provide the following functions:

1) The time required to stop the processor on an error is model-dependent, but must be as immediate as possible.
2) Processor clocks may but need not be stopped by the Stop on Error function. However, the PIT and SIT shall be stopped.
3) Processor restart will perform a process retry as provided by model-dependent hardware.
4) The occurrence of any asynchronous interrupts shall be detected and the appropriate bit set in the MCR on restart of the processor. The processor may be restarted to perform either a retry or higher level recovery via operating system monitor. It is a hardware responsibility to ensure that all asynchronous events are recognized and set in the MCR for the restarted process.

A halt condition that cannot be restarted may be implemented. If it exists in the system, however, this feature normally shall be disabled. This feature shall be selected as a last resort for problem resolution. Usage of this type of a halt condition guarantees an interrupt on error. This is not a fault tolerant design technique, but has an application where troublesome problems require the most accurate system state information at the occurrence of a failure in order to isolate the problem.

Along with the Halt status signal, each element should report how it arrived at the halt condition.

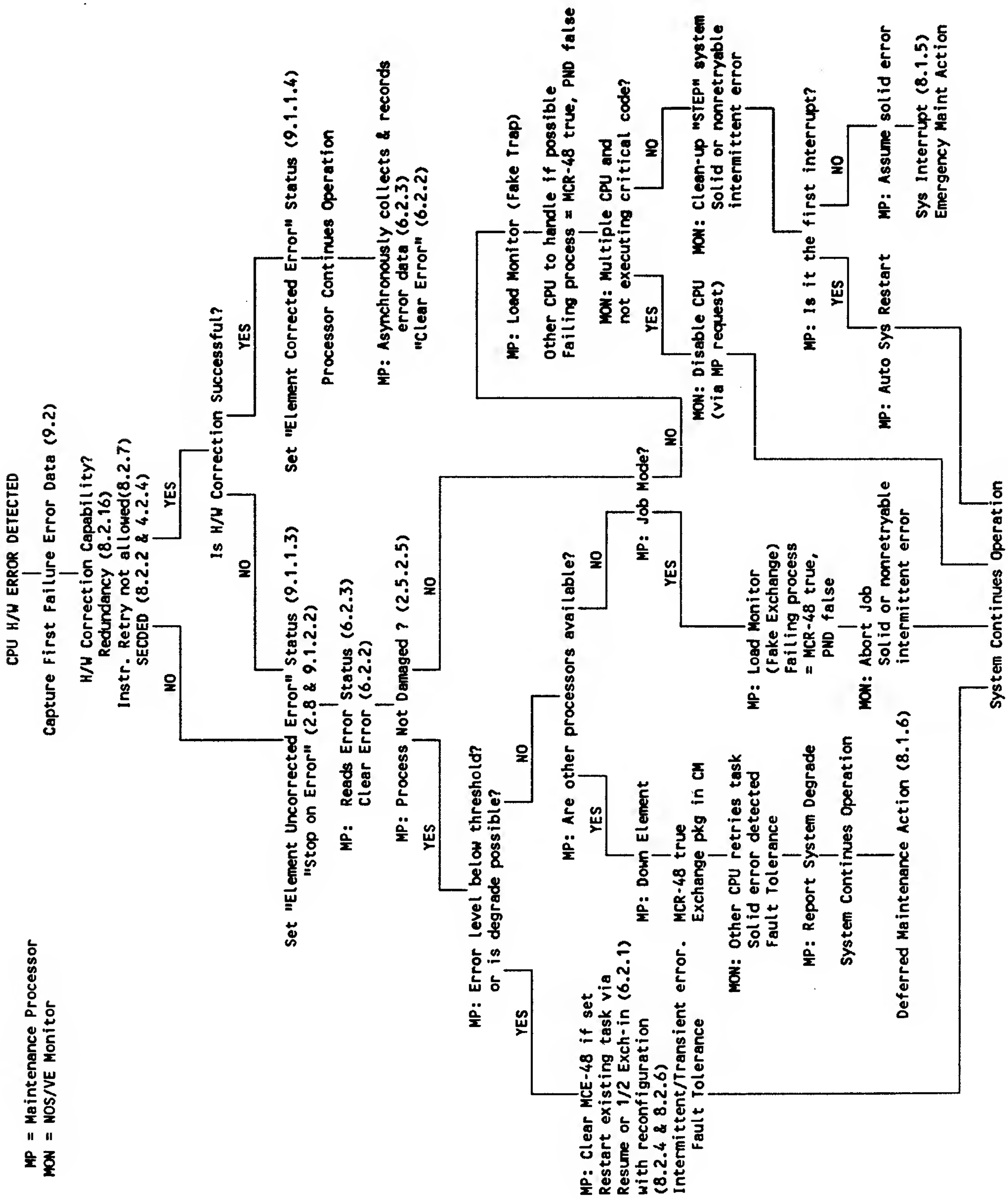


Figure 9.1-1. CPU Error Recovery with CPU Stop on Error

9.1.2.3 Operating State and Mode

The processor element shall report the operating state (C170/C180) and the operating mode (Monitor/Program) at the time that the error occurred.

9.1.2.4 Static Configuration

In order for the Maintenance Processor to make the appropriate reconfiguration decisions, the basic options installed (OI) shall be defined as specified in section 1.5.2.

9.1.2.5 Dynamic Configuration

Dynamic configuration is a model-dependent characteristic which shall be available to the Maintenance Processor for each element. All degradeable or redundant features shall be software selectable. Software switches disable the option/feature when set. Both read and write access to this information shall be provided to the maintenance process.

9.1.2.6 Element Identification

The Maintenance Processor shall be able to uniquely identify each element of the mainframe per the EID register as defined in section 2.5.1.6 and 1.5.

9.1.2.7 Processor Environment

The Maintenance Processor shall be able to capture the processor system environment along with the maintenance error information. The processor environment is defined by obtaining access to the exchange package information. See section 6.2.2 for details on Exchange.

9.2 DETAILED ERROR REPORTING

Detailed error reporting is applicable to all detected corrected and uncorrected errors. Multiple error occurrences may be reported as a single entry, but shall contain a multiple occurrence flag/indication. In shared system resources, the detected error data shall be associated with the using element or path.

Error reporting is necessary to satisfy the record and analysis requirements for a good fault tolerant design. For uncorrected errors, the reported information is necessary to allow the Maintenance Processor the opportunity to recover the operation, assess the impact of the failure, and identify the correct repair action when maintenance is required. The resultant maintenance action may be either immediate or deferred. For corrected errors, the reported information is analyzed to predict possible failure trends or operational states that have exceeded the design limits, thereby requiring repair.

In order to satisfy these objectives the error report shall provide adequate information for the following intended usages:

- Field Replaceable Unit (FRU) Identification - information reported can be manipulated into concise directive for eventual repair.
- FRU Repair - information reported can be sent back with FRU to a repair facility where it can be used to identify one or more replaceable components on that FRU.
- Design Fault Identification - The emphasis here is on isolation of system state at the time that a detection mechanism had detected a hardware or program error. The actual cause or fault condition may have been in software. Where possible the system design shall capture the system state at the time that the error was detected, including information on prior instruction execution history.

Error reporting is generically referred to as "First Error Capture". The components of "First Error Capture" include on-line detection, containment and isolation. Each of these components is model-dependent, because effective placement of detection circuitry is dependent on the hardware architecture.

Since error reporting requirements are model-dependent, detailed information on this subject shall be found in the appropriate engineering specification. The following topics shall be covered in these specifications:

- Status Summary (SS) - each element shall provide a snap-shot of maintenance information which includes the applicable information specified in section 9.1.1.
- Fault Status (FS) - detailed error information provided by each intelligent element.
- Fault Injection - provides the ability to test the system fault tolerant hardware and software implemented to handle errors detected within that element.
- Options Installed (OI) - specifies the element's physical configuration (set on installation).
- Environment Control (EC) - specifies the element's logical configuration (dynamic).
- Corrected Error Log (CEL) - detailed corrected error information for a memory element.
- Uncorrected Error Log (UEL) - detailed uncorrected error information for a memory element.

In the interest of commonality, different maintenance arhitectures providing equivalent data shall follow consistent naming terminology.

APPENDIX A

Ref.	Instruction Name	Op. Code	Mnemonic	Page
001	Load Bytes, Immediate	DSjkiD	LBYTS, S (S=0-7)	2-11
002				
003	Store Bytes, Immediate	DSjkiD	SBYTS, S (S=8-F)	2-11
004				
005	Load Word, Indexed	A2jkiD	LXI	2-12
006	Load Word	82jkQ	LX	2-12
007	Store Word, Indexed	A3jkiD	SXI	2-12
008	Store Word	83jkQ	SX	2-12
009	Load Bytes	A4jkiD	LBYT, X0	2-13
010				
011	Store Bytes	A5jkiD	SBYT, X0	2-13
012				
013	Load Bytes, Relative	86jkQ	LBYTP, j	2-13
014	Load Bit	88jkQ	LBIT	2-14
015	Store Bit	89jkQ	SBIT	2-14
016	Load Address, Indexed	A0jkiD	LAI	2-15
017	Load Address	84jkQ	LA	2-15
018	Store Address, Indexed	A1jkiD	SAI	2-15
019	Store Address	85jkQ	SA	2-15
020	Load Multiple	80jkQ	LMULT	2-16
021	Store Multiple	81jkQ	SMULT	2-16
022	Integer Sum	24jk	ADDX	2-20
023	Integer Difference	25jk	SUBX	2-20
024	Integer Product	26jk	MULX	2-21
025	Integer Quotient	27jk	DIVX	2-21
026				
027	Half Word Integer Sum	20jk	ADDR	2-22
028	Half Word Integer Sum, Signed Immediate	8AjkQ	ADDRQ	2-22
029	Half Word Integer Sum, Immediate	28jk	INCR	2-22
030	Half Word Integer Difference	21jk	SUBR	2-22
031	Half Word Integer Difference, Immediate	29jk	DECR	2-22
032	Half Word Integer Product	22jk	MULR	2-23
033	Half Word Integer Product, Signed Immediate	8CjkQ	MULRQ	2-23
034	Half Word Integer Quotient	23jk	DIVR	2-23
035	Integer Compare	2Djk	CMPX	2-24
036	Half Word Integer Compare	2Cjk	CMPR	2-24
037	Branch on Equal	94jkQ	BRXEQ	2-25
038	Branch on Not Equal	95jkQ	BRXNE	2-25
039	Branch on Greater Than	96jkQ	BRXGT	2-25
040	Branch on Greater Than or Equal	97jkQ	BRXGE	2-25

Ref.	Instruction Name	Op. Code	Mnemonic	Page
041	Branch on Half Word Equal	90jkQ	BRREQ	2-25
042	Branch on Half Word Not Equal	91jkQ	BRRNE	2-25
043	Branch on Half Word Greater Than	92jkQ	BRRGT	2-25
044	Branch on Half Word Greater Than or Equal	93jkQ	BRRGE	2-25
045	Branch and Increment	9CjkQ	BRINC	2-26
046	Branch on Segments Unequal	9DjkQ	BRSEG	2-26
047	Branch Relative	2Ejk	BRREL	2-27
048	Inter-Segment Branch	2Fjk	BRDIR	2-27
049	Copy Full Word	0Djk	CPYXX	2-28
050	Copy Address, A to X	0Bjk	CPYAX	2-28
051	Copy Address, A to A	09jk	CPYAA	2-28
052	Copy Address, X to A	0Ajk	CPYXA	2-28
053	Copy Half Word	0Cjk	CPYRR	2-28
054	Address Increment, Signed Immediate	8EjkQ	ADDAQ	2-29
055	Address Relative	8FjkQ	ADDPXQ	2-29
056	Address Increment, Indexed	2Ajk	ADDAX	2-29
057	Enter Immediate Positive	3Djk	ENTP	2-30
058	Enter Immediate Negative	3Ejk	ENTN	2-30
059	Enter Xk Signed Immediate	8DjkQ	ENTE	2-30
060	Enter X0, Immediate Logical	3Fjk	ENTL	2-31
061	Enter Zeros	1Fjk	ENTZ	2-31
061	Enter Ones	1Fjk	ENTO	2-31
061	Enter Signs	1Fjk	ENTS	2-31
062	Shift Word Circular	A8jkiD	SHFC	2-33
063	Shift Word End-off	A9jkiD	SHFX	2-33
064	Shift Half Word End-off	AAjkiD	SHFR	2-33
065	Logical Sum	18jk	IORX	2-34
066	Logical Difference	19jk	XORX	2-34
067	Logical Product	1Ajk	ANDX	2-34
068	Logical Complement	1Bjk	NOTX	2-34
069	Logical Inhibit	1Cjk	INHX	2-35
070	Isolate Bit Mask	ACjkiD	ISOM	2-36
071	Isolate	ADjkiD	ISOB	2-36
072	Insert	AEjkiD	INSB	2-36
073				
074	Decimal Sum	70jk(2)	ADDN, Aj, X0	2-47
075	Decimal Difference	71jk(2)	SUBN, Aj, X0	2-47
076	Decimal Product	72jk(2)	MULN, Aj, X0	2-47
077	Decimal Quotient	73jk(2)	DIVN, Aj, X0	2-47
078	Decimal Scale	E4jkiD(2)	SCLN, Aj, X0	2-49
079	Decimal Scale, Rounded	E5jkiD(2)	SCLR, Aj, X0	2-49
080				
081				
082				

Ref.	Instruction Name	Op. Code	Mnemonic	Page
083	Decimal Compare	74jk(2)	CMPN, Aj, X0	2-52
084	Byte Compare	77jk(2)	CMPB, Aj, X0	2-52
085	Byte Compare, Collated	E9jkiD(2)	CMPC, Aj, X0	2-52
086	Byte Scan While Nonmember	F3jkiD(1)	SCNB, X0	2-54
087				
088	Byte Translate	EBjkiD(2)	TRANB, Aj, X0	2-54
089	Move Bytes	76jk(2)	MOVB, Aj, X0	2-55
090				
091	Edit	EDjkiD(2)	EDIT, Aj, X0	2-55
092	Numeric Move	75jk(2)	MOVN, Aj, X0	2-51
093				
094				
095				
096				
097	Convert from Integer to Floating Point	3Ajk	CNIF	2-71
098	Convert from Floating Point to Integer	3Bjk	CNFI	2-72
099	Floating Point Sum	30jk	ADDF	2-73
100	Floating Point Difference	31jk	SUBF	2-73
101				
102				
103	Floating Point Product	32jk	MULF	2-76
104	Floating Point Quotient	33jk	DIVF	2-77
105	Double Precision Floating Point Sum	34jk	ADDD	2-79
106	Double Precision Floating Point Difference	35jk	SUBD	2-79
107	Double Precision Floating Point Product	36jk	MULD	2-82
108	Double Precision Floating Point Quotient	37jk	DIVD	2-84
109	Floating Point Branch on Equal	98jkQ	BRFEQ	2-87
110	Floating Point Branch on Not Equal	99jkQ	BRFNE	2-87
111	Floating Point Branch on Greater Than	9AjkQ	BRFGT	2-87
112	Floating Point Branch on Greater Than or Equal	9BjkQ	BRFGE	2-87
113	Floating Point Branch on Overflow	9EjkQ	BROVR	2-88
	Floating Point Branch on Underflow	9EjkQ	BRUND	2-88
	Floating Point Branch on Indefinite	9EjkQ	BRINF	2-88
114	Floating Point Compare	3Cjk	CMPF	2-89

Ref.	Instruction Name	Op. Code	Mnemonic	Page
115	Call Indirect	B5jkQ	CALLSEG	2-122
116	Call Relative	B0jkQ	CALLREL	2-125
117	Return	04jk	RETURN	2-127
118	Pop	06jk	POP	2-129
119				
120	Exchange	02jk	EXCHANGE	2-132
121	Program Error	00jk	HALT	2-122
122	Processor Interrupt	03jk	INTRUPT	2-141
123				
124	Test and Set Bit	14jk	LBSET	2-136
125	Compare Swap	B4jkQ	CMPXA	2-134
126	Test and Set Page	16jk	TPAGE	2-137
127	Load Page Table Index	17jk	LPAGE	2-139
128				
129				
130	Copy from State Register	0Ejk	CPYSX	2-146
131	Copy to State Register	0Fjk	CPYXS	2-146
132	Copy Free Running Counter	08jk	CPYTX	2-137
133				
134	Branch on Condition Register	9FjkQ	BRCR	2-142
135				
136	Keypoint	B1jkQ	KEYPOINT	2-133
137				
138	Purge Buffer	05jk	PURGE	2-147
139	Execute Algorithm	CSjkiD	EXECUTE, S (S=0-7)	2-137
.				
.				
143	Integer, Signed Immediate	8BjkQ	ADDXQ	2-20
144				
145	Mark to Boolean	1Ejk	MARK	2-37
.				
.				
154	Move Immediate Data	F9jkiD(1)	MOVI, Xi, D	2-62
155	Compare Immediate Data	FAjkiD(1)	CMPI, Xi, D	2-63
156	Add Immediate Data	FBjkiD(1)	ADDI, Xi, D	2-64
.				
.				
161	Address Increment, Modulo	A7jkiD	ADDAD	2-30
162				
163				
164	Enter X1, Immediate Logical	39jk	ENTX	2-31
165	Enter X1, Signed Immediate	87jkQ	ENTC	2-31
166	Integer Sum, Immediate	10jk	INCX	2-20
167	Integer Difference, Immediate	11jk	DECX	2-20
168	Integer Product, Signed Immediate	B2jkQ	MULXQ	2-21
169	Enter X0, Signed Immediate	B3jkQ	ENTA	2-31

Ref.	Instruction Name	Op. Code	Mnemonic	Page
170	(Reserved Op. Code)	BEjkQ		2-138
171	(Reserved Op. Code)	BFjkQ		2-138
172	Integer Vector Sum	44jkiD	ADDXV	2-207
173	Integer Vector Difference	45jkiD	SUBXV	2-207
174				
175				
176	Integer Vector Compare, Equal	50jkiD	CMPEQV	2-207
177	Integer Vector Compare, Less Than	51jkiD	CMPLTV	2-207
178	Integer Vector Compare, Greater Than or Equal	52jkiD	CMPGEV	2-207
179	Integer Vector Compare, Not Equal	53jkiD	CMPNEV	2-207
180	Shift Vector Circular	4DjkiD	SHFV	2-208
181	Logical Vector Sum	48jkiD	IORV	2-209
182	Logical Vector Difference	49jkiD	XORV	2-209
183	Logical Vector Product	4AjkiD	ANDV	2-209
184	Convert Vector from Integer to Floating Point	4BjkiD	CNIFV	2-209
185	Convert Vector from Floating Point to Integer	4CjkiD	CNFIV	2-209
186	Floating Point Vector Sum	40jkiD	ADDFV	2-209
187	Floating Point Vector Difference	41jkiD	SUBFV	2-209
188	Floating Point Vector Product	42jkiD	MULFV	2-209
189	Floating Point Vector Quotient	43jkiD	DIVFV	2-209
190	Floating Point Vector Summation	57jkiD	SUMFV	2-210
191	Merge Vector	54jkiD	MRGV	2-210
192	Gather Vector	55jkiD	GTHV	2-211
193	Scatter Vector	56jkiD	SCTV	2-211
194	Scope Loop Sync	01jk	SYNC	2-138
195	Floating Point Vector Triad, * +	58jkiD	TPSFV	2-216
196	Floating Point Vector Triad, * -	59jkiD	TPDFV	2-216
197	Floating Point Vector Triad, + *	5AjkiD	TSPFV	2-216
198	Floating Point Vector Triad, - *	5BjkiD	TDPFV	2-216
199	Floating Point Vector Dot Product	5CjkiD	SUMPFV	2-216
200	Gather Vector, Index	5DjkiD	GTHIV	2-217
201	Scatter Vector, Index	5EjkiD	SCTIV	2-217
202	(Reserved Op. Code)	BDjkQ		2-138
203	Purge SFSA Pushdown	07jk	PSFSA	2-138

APPENDIX B

FORMAT: jk

Op.	Instruction Name	Mnemonic	Address	Page
00	Program Error	HALT	(blank)	2-122
01	Scope Loop Sync	SYNC	(blank)	2-138
02	Exchange	EXCHANGE	(blank)	2-132
03	Processor Interrupt	INTRUPT	Xk	2-141
04	Return	RETURN	(blank)	2-127
05	Purge Buffer	PURGE	Xj, k	2-147
06	Pop	POP	(blank)	2-129
07	Purge SFSA Pushdown	PSFSA	(blank)	2-138
08	Copy Free Running Counter	CPYTX	Xk, Xj	2-137
09	Copy Address, A to A	CPYAA	Ak, Aj	2-28
0A	Copy Address, X to A	CPYXA	Ak, Xj	2-28
0B	Copy Address, A to X	CPYAX	Xk, Aj	2-28
0C	Copy Half Word	CPYRR	Xk, Xj	2-28
0D	Copy Full Word	CPYXX	Xk, Xj	2-28
0E	Copy from State Register	CPYSX	Xk, Xj	2-146
0F	Copy to State Register	CPYXS	Xk, Xj	2-146
10	Integer Sum, Immediate	INCX	Xk, j	2-20
11	Integer Difference, Immediate	DECX	Xk, j	2-20
12				
13				
14	Test and Set Bit	LBSET	Xk, Aj, X0	2-136
15				
16	Test and Set Page	TPAGE	Xk, Aj	2-137
17	Load Page Table Index	LPAGE	Xk, Xj, X1	2-139
18	Logical Sum	IORX	Xk, Xj	2-34
19	Logical Difference	XORX	Xk, Xj	2-34
1A	Logical Product	ANDX	Xk, Xj	2-34
1B	Logical Complement	NOTX	Xk, Xj	2-34
1C	Logical Inhibit	INHX	Xk, Xj	2-35
1D				
1E	Mark to Boolean	MARK	Xk, X1, j	2-37
1F	Enter Zeros/Ones/Signs	ENTZ/O/S	Xk	2-31

FORMAT: jk

Op.	Instruction Name	Mnemonic	Address	Page
20	Half Word Integer Sum	ADDR	Xk, Xj	2-22
21	Half Word Integer Difference	SUBR	Xk, Xj	2-22
22	Half Word Integer Product	MULR	Xk, Xj	2-23
23	Half Word Integer Quotient	DIVR	Xk, Xj	2-23
24	Integer Sum	ADDX	Xk, Xj	2-20
25	Integer Difference	SUBX	Xk, Xj	2-20
26	Integer Product	MULX	Xk, Xj	2-21
27	Integer Quotient	DIVX	Xk, Xj	2-21
28	Half Word Integer Sum, Immediate	INCR	Xk, j	2-22
29	Half Word Integer Difference, Immediate	DECR	Xk, j	2-22
2A	Address Increment,Indexed	ADDAX	Ak, Xj	2-29
2B				
2C	Half Word Integer Compare	CMPR	X1, Xj, Xk	2-24
2D	Integer Compare	CMPX	X1, Xj, Xk	2-24
2E	Branch Relative	BRREL	Xk	2-27
2F	Inter-Segment Branch	BRDIR	Aj, Xk	2-27
30	Floating Point Sum	ADDF	Xk, Xj	2-73
31	Floating Point Difference	SUBF	Xk, Xj	2-73
32	Floating Point Product	MULF	Xk, Xj	2-76
33	Floating Point Quotient	DIVF	Xk, Xj	2-77
34	DP Floating Point Sum	ADDD	XXk, XXj	2-79
35	DP Floating Point Difference	SUBD	XXk, XXj	2-79
36	DP Floating Point Product	MULD	XXk, XXj	2-82
37	DP Floating Point Quotient	DIVD	XXk, XXj	2-84
38				
39	Enter X1, Immediate Logical	ENTX	X1, jk	2-31
3A	Convert from Integer to Floating Point	CNIF	Xk, Xj	2-71
3B	Convert from Floating Point to Integer	CNFI	Xk, Xj	2-72
3C	Floating Point Compare	CMPF	X1, Xj, Xk	2-89
3D	Enter Immediate Positive	ENTP	Xk, j	2-30
3E	Enter Immediate Negative	ENTN	Xk, j	2-30
3F	Enter X0,Immediate Logical	ENTL	X0, jk	2-31

FORMAT: jkiD

Op.	Instruction Name	Mnemonic	Address	Page
40	Floating Point Vector Sum	ADDFV		2-209
41	Floating Point Vector Difference	SUBFV		2-209
42	Floating Point Vector Product	MULFV		2-209
43	Floating Point Vector Quotient	DIVFV		2-209
44	Integer Vector Sum	ADDXV	Ak, Aj, Ai, D	2-207
45	Integer Vector Difference	SUBXV	or	2-207
46			Ak, Xj, Ai, D	
47				
48	Logical Vector Sum	IORV		2-209
49	Logical Vector Difference	XORV		2-209
4A	Logical Vector Product	ANDV		2-209
4B	Convert Vector from Integer to FP	CNIFV	Ak, Aj, D or	2-209
4C	Convert Vector from FP to Integer	CNFIV	Ak, Xj, D	2-209
4D	Shift Vector Circular	SHFV	Ak, Aj, Ai, D	2-208
4E			or	
4F			Ak, Xj, Ai, D	
50	Integer Vector Compare, Equal	COMPEQV ?		2-207
51	Integer Vector Compare, Less Than	CMPLTV	Ak, Aj, Ai, D or	2-207
52	Integer Vector Compare, Greater Than or Equal	CMPGEV	Ak, Xj, Ai, D	2-207
53	Integer Vector Compare, Not Equal	CMPNEV		2-207
54	Merge Vector	MRGV		2-210
55	Gather Vector	GTHV	Ak, Aj, Xi, D	2-210
56	Scatter Vector	SCTV	or Ak, Xj, Xi, D	2-210
57	Floating Point Vector Summation	SUMFV	Xk, Ai, D	2-210
58	Floating Point Vector Triad, * +	TPSFV	Ak, Aj, Ai, X0, D	2-216
59	Floating Point Vector Triad, * -	TPDFV	or	2-216
5A	Floating Point Vector Triad, + *	TSPFV	Ak, Xj, Ai, X0, D	2-216
5B	Floating Point Vector Triad, - *	TDPFV		2-216
5C	Floating Point Vector Dot Product	SUMPFV	Xk, Aj, Ai, D or Xk, Xj, Ai, D	2-216
5D	Gather Vector, Index	GTHIV	Ak, Aj, Ai, D	2-217
5E	Scatter Vector, Index	SCTIV	or Ak, Xj, Ai, D	2-217
5F				

FORMAT: jkiD

Op.	Instruction Name	Mnemonic	Address	Page
60				
.				
.				
.				
6F				

FORMAT: jk(2)

Op.	Instruction Name	Mnemonic	Address	Page
70	Decimal Sum	ADDN, Aj, X0	Ak, X1	2-47
71	Decimal Difference	SUBN, Aj, X0	Ak, X1	2-47
72	Decimal Product	MULN, Aj, X0	Ak, X1	2-47
73	Decimal Quotient	DIVN, Aj, X0	Ak, X1	2-47
74	Decimal Compare	CMPN, Aj, X0	Ak, X1	2-52
75	Numeric Move	MOVN, Aj, X0	Ak, X1	2-51
76	Move Bytes	MOVB, Aj, X0	Ak, X1	2-55
77	Byte Compare	CMPB, Aj, X0	Ak, X1	2-52
78				
79				
7A				
7B				
7C				
7D				
7E				
7F				

FORMAT: jkQ

Op.	Instruction Name	Mnemonic	Address	Page
80	Load Multiple	LMULT	Xk, Aj, Q	2-16
81	Store Multiple	SMULT	Xk, Aj, Q	2-16
82	Load Word	LX	Xk, Aj, Q	2-12
83	Store Word	SX	Xk, Aj, Q	2-12
84	Load Address	LA	Ak, Aj, Q	2-15
85	Store Address	SA	Ak, Aj, Q	2-15
86	Load Bytes, Relative	LBYTP, j	Xk, Q	2-13
87	Enter X1,Signed Immediate	ENTC	X1, jkQ	2-31
88	Load Bit	LBIT	Xk, Aj, Q, X0	2-14
89	Store Bit	SBIT	Xk, Aj, Q, X0	2-14
8A	Half Word Integer Sum, Signed Immediate	ADDRQ	Xk, Xj, Q	2-22
8B	Integer Sum, Signed Immediate	ADDXQ	Xk, Xj, Q	2-20
8C	Half Word Integer Product, Signed Immediate	MULRQ	Xk, Xj, Q	2-23
8D	Enter Xk Signed Immediate	ENTE	Xk, Q	2-30
8E	Address Increment, Signed Immediate	ADDAQ	Ak, Aj, Q	2-29
8F	Address Relative	ADDPXQ	Ak, Xj, Q	2-29
90	Branch on Half Word Equal	BRREQ	Xj, Xk, LABEL	2-25
91	Branch on Half Word Not Equal	BRRNE	Xj, Xk, LABEL	2-25
92	Branch on Half Word Greater Than	BRRGT	Xj, Xk, LABEL	2-25
93	Branch on Half Word Greater Than or Equal	BRRGE	Xj, Xk, LABEL	2-25
94	Branch on Equal	BRXEQ	Xj, Xk, LABEL	2-25
95	Branch on Not Equal	BRXNE	Xj, Xk, LABEL	2-25
96	Branch on Greater Than	BRXGT	Xj, Xk, LABEL	2-25
97	Branch on Greater Than or Equal	BRXGE	Xj, Xk, LABEL	2-25
98	Floating Point Branch on Equal	BRFEQ	Xj, Xk, LABEL	2-87
99	Floating Point Branch on Not Equal	BRFNE	Xj, Xk, LABEL	2-87
9A	Floating Point Branch on Greater Than	BRFGT	Xj, Xk, LABEL	2-87
9B	Floating Point Branch on Greater Than or Equal	BRFGE	Xj, Xk, LABEL	2-87
9C	Branch and Increment	BRINC	Xj, Xk, LABEL	2-26
9D	Branch on Segments Unequal	BRSEG	X1, Aj, Ak, LABEL	2-26
9E	Floating Point Branch on Exception	BR---	Xk, LABEL	2-88
9F	Branch on Condition Register	BRCR	j, k, LABEL	2-142

FORMAT: jkiD

Op.	Instruction Name	Mnemonic	Address	Page
A0	Load Address, Indexed	LAI	Ak, Aj, Xi, D	2-15
A1	Store Address, Indexed	SAI	Ak, Aj, Xi, D	2-15
A2	Load Word, Indexed	LXI	Xk, Aj, Xi, D	2-12
A3	Store Word, Indexed	SXI	Xk, Aj, Xi, D	2-12
A4	Load Bytes	LBYT,X0	Xk, Aj, Xi, D	2-13
A5	Store Bytes	SBYT,X0	Xk, Aj, Xi, D	2-13
A6				
A7	Address Increment, Modulo	ADDAD	Ak, Aj, D, j	2-30
A8	Shift Word Circular	SHFC	Xk, Xj, Xi, D	2-33
A9	Shift Word End-Off	SHFX	Xk, Xj, Xi, D	2-33
AA	Shift Half Word End-off	SHFR	Xk, Xj, Xi, D	2-33
AB				
AC	Isolate Bit Mask	ISOM	Xk, Xi, D	2-36
AD	Isolate	ISOB	Xk, Xj, Xi, D	2-36
AE	Insert	INSB	Xk, Xj, Xi, D	2-36
AF				

FORMAT: jkQ

Op.	Instruction Name	Mnemonic	Address	Page
B0	Call Relative	CALLREL	LABEL, Aj, Ak	2-125
B1	Keypoint	KEYPOINT	j, Xk, Q	2-133
B2	Integer Product, Signed Immediate	MULXQ	Xk, Xj, Q	2-21
B3	Enter X0, Signed Immediate	ENTA	X0, jkQ	2-31
B4	Compare Swap	CMPXA	Xk, Aj, X0, Q	2-134
B5	Call Indirect	CALLSEG	LABEL, Aj, Ak	2-122
B6				
B7				
B8				
B9				
BA				
BB				
BC				
BD	(Reserved Op. Code)			2-138
BE	(Reserved Op. Code)			2-138
BF	(Reserved Op. Code)			2-138

FORMAT: SjkiD

Op.	Instruction Name	Mnemonic	Address	Page
C0	Execute Algorithm 0	EXECUTE, S (S=0-7)	j, k ,i, D	2-137
C1	Execute Algorithm 1			
C2	Execute Algorithm 2			
C3	Execute Algorithm 3			
C4	Execute Algorithm 4			
C5	Execute Algorithm 5			
C6	Execute Algorithm 6			
C7	Exeucte Algorithm 7			
C8				
C9				
CA				
CB				
CC				
CD				
CE				
CF				
D0	Load Bytes, Immediate	LBYTS, S (S=1-8)	Xk, Aj, Xi, D	2-11
D1	Load Bytes, Immediate			
D2	Load Bytes, Immediate			
D4	Load Bytes, Immediate			
D5	Load Bytes, Immediate			
D6	Load Bytes, Immediate			
D7	Load Bytes, Immediate			
D8	Store Bytes, Immediate	SBYTS, S (S=1-8)	Xk, Aj, Xi, D	2-11
D9	Store Bytes, Immediate			
DA	Store Bytes, Immediate			
DB	Store Bytes, Immediate			
DC	Store Bytes, Immediate			
DD	Store Bytes, Immediate			
DE	Store Bytes, Immediate			
DF	Store Bytes, Immediate			

FORMAT: jkiD(2)

Op.	Instruction Name	Mnemonic	Address	Page
E0				
E1				
E2				
E3				
E4	Decimal Scale	SCLN, Aj, X0	Ak, X1, Xi, D	2-49
E5	Decimal Scale, Rounded	SCLR, Aj, X0	Ak, X1, Xi, D	2-49
E6				
E7				
E8				
E9	Byte Compare, Collated	CMPC, Aj, X0	Ak, X1, Ai, D	2-52
EA				
EB	Byte Translate	TRANB, Aj, X0	Ak, X1, Ai, D	2-54
EC				
ED	Edit	EDIT, Aj, X0	Ak, X1, Ai, D	2-55
EE				
EF				

FORMAT: jkiD(1)

Op.	Instruction Name	Mnemonic	Address	Page
F0				
F1				
F2				
F3	Byte Scan While Nonmember	SCNB, X0	Ak, X1, Ai, D	2-54
F4				
F5				
F6				
F7				
F8				
F9	Move Immediate Data	MOVI, Xi, D	Ak, X1, j	2-62
FA	Compare Immediate Data	CMPI, Xi, D	Ak, X1, j	2-63
FB	Add Immediate Data	ADDI, Xi, D	Ak, X1, j	2-64
FC				
FD				
FE				
FF				

APPENDIX C: Edit Examples

Appendix C: Edit Examples
Edit Masks 1 through 25

Mask No.	Cobol Picture	Edit Mask {Hexadecimal with insertion characters *, $, 0, /, b, C and R shown as alphanumerics}
1	$ZZ,ZZ9.99	08 96 72 C4 72 01 95 02
2	$ZZ,ZZZ.99	07 96 72 C4 73 95 02
3	$ZZ,ZZZ.ZZ	08 96 72 C4 73 95 02 FA
4	-ZZZZ9.99	06 B3 74 01 95 02
5	ZZZZ9.99+	07 74 01 95 02 52 98
6	ZZ.999,99	06 72 C5 03 94 02 {Decimal Point is Comma}
7	$$$$.99CR	0B 61 $ 73 80 95 02 62 C R B8
8	$$$,$$$.$$	0A 61 $ 72 C4 73 80 95 02 FA
9	$$$$99,99CR	0C 61 $ 73 80 02 94 02 62 C R B8
10	$$$,$$9.99	0A 61 $ 72 C4 72 80 01 95 02
11	$99.99	05 96 02 95 02
12	$**,**9.99	0A 96 D1 * 72 C4 72 01 95 02
13	$**,***.**BCR	11 96 D1 * 72 94 73 95 02 63 b C R B8 F7 95 E5
14	$**,***.**	0C 96 D1 * 72 C4 73 95 02 F7 95 E2
15	**,***.**+	0D D1 * 72 C4 73 95 02 52 98 F6 95 E3
16	--99999,99	07 50 71 80 05 94 02
17	----.99	06 50 73 80 95 02
18	++++99	05 52 73 80 02
19	00999.00	09 42 0 0 03 95 42 0 0
20	99,999	05 02 C4 03 F6 {Blank When Zero}
21	XX/XX/XX	08 12 41 / 12 41 / 12
22	BBB99.99-	09 43 b b b 02 95 02 B3
23	999.00	06 03 95 42 0 0
24	999.BB	06 03 95 42 b b
25	9B9B9	06 01 91 01 91 01 or 08 01 41 b 01 41 b 01

Appendix C: Edit Examples

Edit Examples 1 through 31 using Edit Masks 1 through 8

Example No.	Source {<u>logical</u> contents}	Mask No.	Destination {result}
1	00000.00	1	$bbbbb0.00
2	00000.01	1	$bbbbb0.01
3	00000.10	1	$bbbbb0.10
4	00001.00	1	$bbbbb1.00
5	00010.00	1	$bbbb10.00
6	00100.00	1	$bbb100.00
7	01000.00	1	$b1,000.00
8	10000.00	1	$10,000.00
9	00000.00	2	$bbbbbb.00
10	00000.00	3	bbbbbbbbbb
11	00000.01	3	$bbbbbb.01
12	00001.00	3	$bbbbb1.00
13	10000.00	3	$10,000.00
14	-00000.00	4	-bbbb0.00
15	+00000.00	4	bbbbb0.00
16	-12345.67	5	12345.67-
17	+00012.34	5	bbb12.34+
18	00000.00	6	bbb000,00
19	01000.00	6	b1.000,00
20	-123.45	7	$123.45CR
21	-023.45	7	B$23.45CR
22	003.45	7	bb$3.45bb
23	000.45	7	bbb$.45bb
24	00000.00	8	bbbbbbbbbb
25	00000.01	8	bbbbbb$.01
26	00000.10	8	bbbbbb$.10
27	00001.00	8	bbbbb$1.00
28	00010.00	8	bbbb$10.00
29	00100.00	8	bbb$100.00
30	01000.00	8	b$1,000.00
31	10000.00	8	$10,000.00

Appendix C: Edit Examples
Edit Examples 32 through 61 using Edit Masks 9 through 16

Example No.	Source {logical contents}	Mask No.	Destination {result}
32	-0000000	9	bbb$00,00CR
33	0010000	9	bb$100,00bb
34	0100000	9	b$1000,00bb
35	-1000000	9	$10000,00CR
36	00000.00	10	bbbbb$0.00
37	10000.00	10	$10,000.00
38	00.00	11	$00.00
39	12.34	11	$12.34
40	00000.00	12	$*****0.00
41	00000.01	12	$*****0.01
42	00000.10	12	$*****0.10
43	00001.00	12	$*****1.00
44	00010.00	12	$****10.00
45	00100.00	12	$***100.00
46	01000.00	12	$*1,000.00
47	10000.00	12	$10,000.00
48	00000.00	13	*******.*****
49	-00000.01	13	$**,***.01bCR
50	00000.01	13	$**,***.01bbb
51	-00000.00	13	*******.*****
52	00000.00	14	*******.**
53	-00000.01	14	$******.01
54	00000.00	15	******.***
55	-00000.00	15	******.***
56	12345.67	15	12,345.67*
57	-12345.67	15	12,345.67-
58	-00000000	16	b-00000,00
59	-12345678	16	-123456,78
60	00000000	16	bb00000,00
61	12345678	16	b123456,78

Appendix C: Edit Examples
Edit Examples 62 through 89 using Edit Masks 17 through 25

Example No.	Source {logical contents}	Mask No.	Destination {result}
62	-000.00	17	bbb-.00
63	000.00	17	bbbb.00
64	-001.00	17	bb-1.00
65	010.00	17	bb10.00
66	-100.00	17	-100.00
67	00000	18	bbb+00
68	-00000	18	bbb-00
69	00012	18	bbb+12
70	-00123	18	bb-123
71	01234	18	b+1234
72	-12345	18	-12345
73	000	19	00000.00
74	-123	19	00123.00
75	123	19	00123.00
76	00000	20	bbbbbb
77	00001	20	00,001
78	HHMMSS	21	HH/MM/SS
79	-00.00	22	bbb00.00-
80	00.00	22	bbb00.00b
81	12.34	22	bbb12.34b
82	000	23	000.00
83	-123	23	123.00
84	123	23	123.00
85	000	24	000.bb
86	-123	24	123.bb
87	123	24	123.bb
88	000	25	0b0b0
89	123	25	1b2b3

Appendix C: Edit Examples
Edit Examples 90 and 91 using Edit Mask 26

Edit Mask No. 26

Cobol
Picture: $ $ $ $, $ $ $, $ $ $, $ $ $. $ $ $, $ $ $

Edit Mask: 11 61 $ 73 C4 73 C4 73 C4 73 80 95 03 94 03 FF E9

Example No. 90 using Edit Mask No. 26

Source
(logical contents): 0 0 0 0 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0

Destination
(result): b b b b b b b b b b b b b b b b b b b b b b b b

Example No. 91 using Edit Mask No. 26

Source
(logical contents): 1 2 3 4 5 6 7 8 9 0 1 2 . 6 5 4 3 2 1

Destination
(result): $ 1 2 3 , 4 5 6 , 7 8 9 , 0 1 2 . 6 5 4 , 3 2 1

Note: For the examples in this appendix, the destination field is assumed to have the same length and decimal point position as the source field except for the differences necessitated by insertion characters.

APPENDIX D: Interrupt Conditions

INTERRUPT CONDITIONS - 0X

jk INSTRUCTION FORMAT

0	8	12 15
OP	j	k

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR. * MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR. * MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG * UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
00	HALT	PROGRAM ERROR	X	X																				
01	SYNC	SCOPE LOOP SYNC	X																					
02	EXCHANGE	EXCHANGE	X				X		X															
03	INTRUPT	PROCESSOR INTERRUPT	X									X												
04	RETURN	RETURN	X		X	X	X	X		X	X				X		X							
05	PURGE	PURGE BUFFER	X		X					X		X												
06	POP	POP	X		X	X	X	X		X				X	X		X							
07		PURGE SPSA PUSHDOWN	X																					
08	CPYTX	COPY FREE RUNNING COUNTER	X																					
09	CPYAA	COPY ADDRESS A TO A	X																					
0A	CPYXA	COPY ADDRESS X TO A	X																					
0B	CPYAX	COPY ADDRESS A TO X	X																					
0C	CPYRR	COPY HALF WORD	X																					
0D	CPYXX	COPY FULL WORD	X																					
0E	CPYSX	COPY FROM STATE REGISTER	X																					
0F	CPYXS	COPY TO STATE REGISTER	X	X								X												

Conditions which may result from any instruction fetch are marked with an "*".
Conditions which may result from the first instruction fetch following an exchange are marked with a "●".
Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 1X

jk INSTRUCTION FORMAT

0	8	12 15
OP	j	k

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR. * MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR. * MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG * UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
10	INCX	INTEGER SUM IMMEDIATE	X															X						
11	DECX	INTEGER DIFFERENCE IMM.	X															X						
12													X											
13													X											
14	LBSET	TEST AND SET BIT	X		X	X		X		X							X							
15													X											
16	TPAGE	TEST AND SET PAGE	X		X					X														
17	LPAGE	LOAD PAGE TABLE INDEX	X		X							X												
18	IORX	LOGICAL SUM	X																					
19	XORX	LOGICAL DIFFERENCE	X																					
1A	ANDX	LOGICAL PRODUCT	X																					
1B	NOTX	LOGICAL COMPLEMENT	X																					
1C	INHX	LOGICAL INHIBIT	X																					
1D													X											
1E	MARK	MARK TO BOOLEAN	X																					
1F	ENT-	ENTER ZEROS/ONES/SIGNS	X																					

Conditions which may result from any instruction fetch are marked with an "*".
Conditions which may result from the first instruction fetch following an exchange are marked with a "●".
Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 2X

jk INSTRUCTION FORMAT

0 OP	8 j	12 k 15

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR.* MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR.* MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG* UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
20	ADDR	HALF WORD INTEGER SUM	X															X						
21	SUBR	HALF WORD INTEGER DIFF.	X															X						
22	MULR	HALF WORD INTEGER PROD.	X															X						
23	DIVR	HALF WORD INTEGER QUOT.	X													X		X						
24	ADDX	INTEGER SUM	X															X						
25	SUBX	INTEGER DIFFERENCE	X															X						
26	MULX	INTEGER PRODUCT	X															X						
27	DIVX	INTEGER QUOTIENT	X													X		X						
28	INCR	HALF WORD INT. SUM IMM.	X															X						
29	DECR	HALF WORD INT. DIFF. IMM.	X															X						
2A	ADDAX	ADDRESS INCREMENT INDEXED	X																					
2B													X											
2C	CMPR	HALF WORD INT. COMPARE	X																					
2D	CMPX	INTEGER COMPARE	X																					
2E	BRREL	BRANCH RELATIVE	X		X			X									X							
2F	BRDIR	INTER-SEGMENT BRANCH	X		X	X		X		X							X							

Conditions which may result from any instruction fetch are marked with an "*".
Conditions which may result from the first instruction fetch foilowing an exchange are marked with a "●".
Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 3X

jk INSTRUCTION FORMAT

0 OP	8 j	12 k 15

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR.* MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR.* MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR55	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG* UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
30	ADDF	SINGLE PRECISION FP SUM	X																X	X	X	X		
31	SUBF	SINGLE PRECISION FP DIFF.	X																X	X	X	X		
32	MULF	SINGLE PRECISION FP PROD.	X																X	X		X		
33	DIVF	SINGLE PRECISION FP QUOT.	X													X			X	X		X		
34	ADDD	DOUBLE PRECISION FP SUM	X																X	X	X	X		
35	SUBD	DOUBLE PRECISION FP DIFF.	X																X	X	X	X		
36	MULD	DOUBLE PRECISION FP PROD.	X																X	X		X		
37	DIVD	DOUBLE PRECISION FP QUOT.	X													X			X	X		X		
38													X											
39	ENTX	ENTER X1 IMM. LOGICAL	X																					
3A	CNIF	CONVERT FROM INT. TO FP	X																					
3B	CNFI	CONVERT FROM FP TO INT.	X																			X	X	
3C	CMPF	FLOATING POINT COMPARE	X																			X		
3D	ENTP	ENTER IMMEDIATE POSITIVE	X																					
3E	ENTN	ENTER IMMEDIATE NEGATIVE	X																					
3F	ENTL	ENTER X0 IMM. LOGICAL	X																					

Conditions which may result from any instruction fetch are marked with an "*".
Conditions which may result from the first instruction fetch following an exchange are marked with a "●".
Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 4X

jkiD INSTRUCTION FORMAT

0	8	12	16	20	31
OP	j	k	i	D	

			INTERRUPT CONDITION																					
OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR.* MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR.* MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG* UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
40	ADDFV	FP VECTOR SUM	X	X	X	X	X	X		X							X		X	X	X	X		
41	SUBFV	FP VECTOR DIFFERENCE	X	X	X	X	X	X		X							X		X	X	X	X		
42	MULFV	FP VECTOR PRODUCT	X	X	X	X	X	X		X							X		X	X		X		
43	DIVFV	FP VECTOR QUOTIENT	X	X	X	X	X	X		X						X	X		X	X		X		
44	ADDXV	INTEGER VECTOR SUM	X	X	X	X	X	X		X							X	X						
45	SUBXV	INTEGER VECTOR DIFF.	X	X	X	X	X	X		X							X	X						
46													X											
47													X											
48	IORV	LOGICAL VECTOR SUM	X	X	X	X	X	X		X							X							
49	XORV	LOGICAL VECTOR DIFF.	X	X	X	X	X	X		X							X							
4A	ANDV	LOGICAL VECTOR PROD.	X	X	X	X	X	X		X							X							
4B	CNIFV	CONVERT VECTOR FROM INT. TO FP	X	X	X	X	X	X		X							X							
4C	CNFIV	CONVERT VECTOR FROM FP TO INT.	X	X	X	X	X	X		X							X					X	X	
4D	SHFV	SHIFT VECTOR CIRCULAR	X	X	X	X	X	X		X							X							
4E													X											
4F													X											

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 5X, 6X

jkiD INSTRUCTION FORMAT

0	8	12	16	20	31
OP	j	k	i	D	

			INTERRUPT CONDITION																					
OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR.* MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR.* MCR52	ACCESS VIOLATION ● MCR54	ENV. SPC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVID FAULT UCR55	DEBUG* UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
50	CMPEQV	INTEGER VECTOR COMPARE =	X	X	X	X	X	X		X							X							
51	CMPLTV	INTEGER VECTOR COMPARE <	X	X	X	X	X	X		X							X							
52	CMPGEV	INTEGER VECTOR COMPARE >	X	X	X	X	X	X		X							X							
53	CMPNEV	INTEGER VECTOR COMPARE ≠	X	X	X	X	X	X		X							X							
54	MRGV	MERGE VECTOR	X	X	X	X	X	X		X							X							
55	GTHV	GATHER VECTOR	X	X	X	X	X	X		X							X							
56	SCTV	SCATTER VECTOR	X	X	X	X	X	X		X							X							
57	SUMFV	FP VECTOR SUMMATION	X	X	X	X	X	X		X							X		X	X	X	X		
58	TPSFV	FP VECTOR TRIAD, *+	X	X	X	X	X	X		X							X		X	X	X	X		
59	TPDFV	FP VECTOR TRIAD, *-	X	X	X	X	X	X		X							X		X	X	X	X		
5A	TSPFV	FP VECTOR TRIAD, +*	X	X	X	X	X	X		X							X		X	X	X	X		
5B	TDPFV	FP VECTOR TRIAD, -*	X	X	X	X	X	X		X							X		X	X	X	X		
5C	SUMPFV	FP VECTOR DOT PRODUCT	X	X	X	X	X	X		X							X		X	X	X	X		
5D	GTHJIV	GATHER VECTOR, INDEX	X	X	X	X	X	X		X							X							
5E	SCTIV	SCATTER VECTOR, INDEX	X	X	X	X	X	X		X							X							
5F													X											
60 ↓ 6F													X ↓											

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 7X

jk(2) INSTRUCTION FORMAT

0 OP	8 j	12 k 15

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR. * MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR. * MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WD FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG * UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
70	ADDN, Aj, X0	DECIMAL SUM	X	X	X	X		X		X							X	X						X
71	SUBN, Aj, X0	DECIMAL DIFFERENCE	X	X	X	X		X		X							X	X						X
72	MULN, Aj, X0	DECIMAL PRODUCT	X	X	X	X		X		X							X	X						X
73	DIVN, Aj, X0	DECIMAL QUOTIENT	X	X	X	X		X		X						X	X	X						X
74	CMPN, Aj, X0	DECIMAL COMPARE	X	X	X	X		X		X							X							X
75	MOVN, Aj, X0	DECIMAL MOVE	X	X	X	X		X		X							X						X	X
76	MOVB, Aj, X0	MOVE BYTES	X	X	X	X		X		X							X							
77	CMPB, Aj, X0	BYTE COMPARE	X	X	X	X		X		X							X							
78													X											
79													X											
7A													X											
7B													X											
7C													X											
7D													X											
7E													X											
7F													X											

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 8X

jkQ INSTRUCTION FORMAT

0 OP	8 j	12 k	16 Q 31

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR. * MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR. * MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WD FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR61	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG * UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
80	LMULT	LOAD MULTIPLE	X		X	X		X		X							X							
81	SMULT	STORE MULTIPLE	X		X	X		X		X							X							
82	LX	LOAD WORD	X		X	X		X		X							X							
83	SX	STORE WORD	X		X	X		X		X							X							
84	LA	LOAD ADDRESS	X		X	X		X		X							X							
85	SA	STORE ADDRESS	X		X	X		X		X							X							
86	LBYTP, j	LOAD BYTES RELATIVE	X		X	X		X		X							X							
87	ENTC	ENTER X1 SIGNED IMM.	X																					
88	LBIT	LOAD BIT	X		X	X		X		X							X							
89	SBIT	STORE BIT	X		X	X		X		X							X							
8A	ADDRQ	HALF WD INTEGER SUM SIGNED IMM.	X															X						
8B	ADDXQ	INTEGER SUM SIGNED IMM.	X															X						
8C	MULRQ	HALF WD INTEGER PRDD. SIGNED IMM.	X															X						
8D	ENTE	ENTER SIGNED IMMEDIATE	X																					
8E	ADDAQ	ADDRESS INCR. SIGNED IMM.	X																					
8F	ADDPXQ	ADDRESS RELATIVE	X																					

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - 9X

jkQ INSTRUCTION FORMAT

0 OP	8 j	12 k	16 Q 31

OP	MNEMONIC	INSTRUCTION NAME	MCR48 DET. UNCORR. ERR. *	MCR51 INST. SPEC. ERR.	MCR52 ADDR. SPEC. ERR. *	MCR54 ACCESS VIOLATION ●	MCR55 ENV. SPEC. ERR.	MCR57 PT SEARCH WO FIND*	MCR58 SYSTEM CALL	MCR60 INV. SEG/RN 0 ●	MCR61 OUT. CALL/IN. RET.	UCR48 PRIV. INST. FAULT	UCR49 UNIMPL'D. INST.	UCR52 INTER-RING POP	UCR53 CRIT. FRAME FLAG	UCR55 DIVIDE FAULT	UCR56 DEBUG *	UCR57 ARITH. OVERFLOW	UCR58 EXP. OVERFLOW	UCR59 EXP. UNDERFLOW	UCR60 FP LOSS OF SIG.	UCR61 FP INDEFINITE	UCR62 ARITH. LOSS OF SIG.	UCR63 INV. BDP DATA
90	BRREQ	BRANCH ON HALF WORD =	X		X			X									X							
91	BRRNE	BRANCH ON HALF WORD ≠	X		X			X									X							
92	BRRGT	BRANCH ON HALF WORD >	X		X			X									X							
93	BRRGE	BRANCH ON HALF WORD ≥	X		X			X									X							
94	BRXEQ	BRANCH ON =	X		X			X									X							
95	BRXNE	BRANCH ON ≠	X		X			X									X							
96	BRXGT	BRANCH ON >	X		X			X									X							
97	BRXGE	BRANCH ON ≥	X		X			X									X							
98	BRFEQ	FP BRANCH ON =	X		X			X									X					X		
99	BRFNE	FP BRANCH ON ≠	X		X			X									X					X		
9A	BRFGT	FP BRANCH ON >	X		X			X									X					X		
9B	BRFGE	FP BRANCH ON ≥	X		X			X									X					X		
9C	BRINC	BRANCH AND INCREMENT	X		X			X									X							
9D	BRSEG	BRANCH ON SEGMENTS UNEQUAL	X		X			X									X							
9E	BR---	FP BRANCH ON EXCEPTION	X		X			X									X							
9F	BRCR	BRANCH ON CONDITION REG.(See 1.6)	X	X	X			X									X							

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - AX

jkiD INSTRUCTION FORMAT

0 OP	8 j	12 k	16 i	20 D 31

OP	MNEMONIC	INSTRUCTION NAME	MCR48 DET. UNCORR. ERR. *	MCR51 INST. SPEC. ERR.	MCR52 ADDR. SPEC. ERR. *	MCR54 ACCESS VIOLATION ●	MCR55 ENV. SPEC. ERR.	MCR57 PT SEARCH WO FIND*	MCR58 SYSTEM CALL	MCR60 INV. SEG/RN 0 ●	MCR61 OUT. CALL/IN. RET.	UCR48 PRIV. INST. FAULT	UCR49 UNIMPL'D. INST.	UCR52 INTER-RING POP	UCR53 CRIT. FRAME FLAG	UCR55 DIVIDE FAULT	UCR58 DEBUG *	UCR57 ARITH. OVERFLOW	UCR58 EXP. OVERFLOW	UCR59 EXP. UNDERFLOW	UCR60 FP LOSS OF SIG.	UCR61 FP INDEFINITE	UCR62 ARITH. LOSS OF SIG.	UCR63 INV. BDP DATA
A0	LAI	LOAD ADDRESS INDEXED	X		X	X		X		X							X							
A1	SAI	STORE ADDRESS INDEXED	X		X	X		X		X							X							
A2	LXI	LOAD WORD INDEXED	X		X	X		X		X							X							
A3	SXI	STORE WORD INDEXED	X		X	X		X		X							X							
A4	LBYT, X0	LOAD BYTES	X		X	X		X		X							X							
A5	SBYT, X0	STORE BYTES	X		X	X		X		X							X							
A6													X											
A7	ADDAD	ADDRESS INCREMENT, MODULO	X																					
A8	SHFC	SHIFT WORD CIRCULAR	X																					
A9	SHFX	SHIFT WORD END-OFF	X																					
AA	SHFR	SHIFT HALF WORD END-OFF	X																					
AB													X											
AC	ISOM	ISOLATE BIT MASK	X	X																				
AD	ISOB	ISOLATE	X	X																				
AE	INSB	INSERT	X	X																				
AF													X											

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - BX

jkQ INSTRUCTION FORMAT

0	8	12	16 31
OP	j	k	Q

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR. * MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR. * MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR55	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR51	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG * UCR58	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR51	ARITH. LOSS OF SIG. UCR82	INV. BOP DATA UCR63
B0	CALLREL	CALL RELATIVE	X	X	X	X		X		X							X							
B1	KEYPOINT	KEYPOINT	X																					
B2	MULXO	INTEGER PRDD. SIGNED IMM.	X															X						
B3	ENTA	ENTER X0 SIGNED IMM.	X																					
B4	CMPXA	COMPARE SWAP	X	X	X	X		X		X							X							
B5	CALLSEG	CALL INDIRECT	X	X	X	X	X	X		X	X						X							
B6													X											
B7													X											
B8													X											
B9													X											
BA													X											
BB													X											
BC													X											
BD		(RESERVED OP CODE)											X											
BE		(RESERVED OP CODE)											X											
BF		(RESERVEO OP CODE)											X											

Conditions which may result from any instruction fetch are marked with an "*".
Conditions which may result from the first instruction fetch following an exchange are marked with a "●".
Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - CX, DX

SjkiD INSTRUCTION FORMAT

0	5	8	12	16	20 31
OP	S	j	k	i	D

OP	MNEMONIC	INSTRUCTION NAME	DET. UNCORR. ERR. * MCR48	INST. SPEC. ERR. MCR51	ADDR. SPEC. ERR. * MCR52	ACCESS VIOLATION ● MCR54	ENV. SPEC. ERR. MCR55	PT SEARCH WO FIND* MCR57	SYSTEM CALL MCR58	INV. SEG/RN 0 ● MCR60	OUT. CALL/IN. RET. MCR51	PRIV. INST. FAULT UCR48	UNIMPL'D. INST. UCR49	INTER-RING POP UCR52	CRIT. FRAME FLAG UCR53	DIVIDE FAULT UCR55	DEBUG * UCR56	ARITH. OVERFLOW UCR57	EXP. OVERFLOW UCR58	EXP. UNDERFLOW UCR59	FP LOSS OF SIG. UCR60	FP INDEFINITE UCR61	ARITH. LOSS OF SIG. UCR62	INV. BDP DATA UCR63
C0 ↓ C7	EXECUTE, S ↓	EXECUTE ALGORITHM 0 ↓ EXECUTE ALGORITHM 7	X ↓																					
C8 ↓ CF													X ↓											
D0 ↓ D7	LBYTS, S ↓	LDAO BYTES, IMMEOIATE ↓	X ↓		X ↓	X ↓		X ↓		X ↓							X ↓							
D8 ↓ DF	SBYTS, S ↓	STORE BYTES, IMMEOIATE ↓	X ↓		X ↓	X ↓		X ↓		X ↓							X ↓							

Conditions which may result from any instruction fetch are marked with an "*".
Conditions which may result from the first instruction fetch following an exchange are marked with a "●".
Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - EX

jkiD(2) INSTRUCTION FORMAT

0 OP	8 j	12 k	16 i	20 D 31

OP	MNEMONIC	INSTRUCTION NAME	MCR48 DET. UNCORR. ERR. *	MCR51 INST. SPEC. ERR.	MCR52 ADDR. SPEC. ERR. *	MCR54 ACCESS VIOLATION ●	MCR55 ENV. SPEC. ERR.	MCR57 PT SEARCH WO FIND *	MCR58 SYSTEM CALL	MCR60 INV. SEG/RN 0 ●	MCR61 OUT. CALL/IN. RET.	UCR48 PRIV. INST. FAULT	UCR49 UNIMPL'D. INST.	UCR52 INTER-RING POP	UCR53 CRIT. FRAME FLAG	UCR55 DIVIDE FAULT	UCR56 DEBUG *	UCR57 ARITH. OVERFLOW	UCR58 EXP. OVERFLOW	UCR59 EXP. UNDERFLOW	UCR60 FP LOSS OF SIG.	UCR61 FP INDEFINITE	UCR62 ARITH. LOSS OF SIG.	UCR63 INV. BDP DATA
E0													X											
E1													X											
E2													X											
E3													X											
E4	SCLN, Aj, X0	DECIMAL SCALE	X	X	X	X		X		X							X						X	X
E5	SCLR, Aj, X0	DECIMAL SCALE ROUNDED	X	X	X	X		X		X							X						X	X
E6													X											
E7													X											
E8													X											
E9	CMPC, Aj, X0	BYTE COMPARE COLLATED	X	X	X	X		X		X							X							
EA													X											
EB	TRANB, Aj, X0	BYTE TRANSLATE	X	X	X	X		X		X							X							
EC													X											
ED	EDIT, Aj, X0	EDIT	X	X	X	X		X		X							X							X
EE													X											
EF													X											

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

INTERRUPT CONDITIONS - FX

jkiD(1) INSTRUCTION FORMAT

0 OP	8 j	12 k	16 i	20 D 31

OP	MNEMONIC	INSTRUCTION NAME	MCR48 DET. UNCORR. ERR. *	MCR51 INST. SPEC. ERR.	MCR52 ADDR. SPEC. ERR. *	MCR54 ACCESS VIOLATION ●	MCR55 ENV. SPEC. ERR.	MCR57 PT SEARCH WO FIND *	MCR58 SYSTEM CALL	MCR60 INV. SEG/RN 0 ●	MCR61 OUT. CALL/IN. RET.	UCR48 PRIV. INST. FAULT	UCR49 UNIMPL'D. INST.	UCR52 INTER-RING POP	UCR53 CRIT. FRAME FLAG	UCR55 DIVIDE FAULT	UCR56 DEBUG *	UCR57 ARITH. OVERFLOW	UCR58 EXP. OVERFLOW	UCR59 EXP. UNDERFLOW	UCR60 FP LOSS OF SIG.	UCR61 FP INDEFINITE	UCR62 ARITH. LOSS OF SIG.	UCR63 INV. BDP DATA
F0													X											
F1													X											
F2													X											
F3	SCNB, X0	BYTE SCAN WHILE NON-MEMBER	X	X	X	X		X		X							X							
F4													X				X							X
F5													X											
F6													X											
F7													X											
F8													X											
F9	MOVI, Xi, D	MOVE IMMEDIATE DATA	X	X	X	X		X		X							X						X	X
FA	CMPI, Xi, D	COMPARE IMMEDIATE DATA	X	X	X	X		X		X							X							X
FB	ADDI, Xi, D	ADD IMMEDIATE DATA	X	X	X	X		X		X							X	X						X
FC													X											
FD													X											
FE													X											
FF													X											

Conditions which may result from any instruction fetch are marked with an "*".

Conditions which may result from the first instruction fetch following an exchange are marked with a "●".

Conditions which may result from instruction execution are marked with an "X".

APPENDIX E: PP Instructions

Code	Description		Page	Code	Description		Page
0000	Pass		5-44	1000d	Central read and set lock from d to (A)		5-35
0001dm	Long jump to m+(d)		5-21	1001d	Central read and clear lock from d to (A)		5-35
0002dm	Return jump to m+(d)		5-21	1002	Pass		5-44
0003d	Unconditional jump d		5-21	1003	Pass		5-44
0004d	Zero jump d		5-21	1004	Pass		5-44
0005d	Nonzero jump d		5-21	1005	Pass		5-44
0006d	Plus jump d		5-21				
0007d	Minus jump d		5-21				
0010d	Shift d		5-13	1010d	Shift (d)	[2]	5-14
0011d	Logical difference d		5-14	1011d	Load R-Register Upper (d)	[2]	5-28
0012d	Logical product d		5-14	1012d	Load R-Register Upper ((d))	[2]	5-28
0013d	Selective clear d		5-15	1013dm	Load R-Register Upper (m+(d))	[2]	5-28
0014d	Load d		5-9	1014d	Store R-Register Upper (d)	[2]	5-28
0015d	Load complement d		5-9	1015d	Store R-Register Upper ((d))	[2]	5-28
0016d	Add d		5-11	1016dm	Store R-Register Upper (m+(d))	[2]	5-28
0017d	Subtract d		5-11	1017	Wait (A)	[2]	5-44
0020dm	Load dm		5-9	1020	Pass		5-44
0021dm	Add dm		5-11	1021	Pass		5-44
0022dm	Logical product dm		5-15	1022d	Logical Product (d) long		5-15
0023dm	Logical difference dm		5-16	1023d	Logical Product ((d)) long		5-15
002400	Pass		5-44	1024dm	Logical Product (A) (m+(d)) long		5-15
002500	Pass		5-44	1025	Pass		5-44
00260x	Exchange jump	[1]	5-44	1026d	Interrupt processor on memory port d		5-46
00261x	Monitor exchange jump	[1]	5-44				
00262x	Monitor exchange jump to MA	[1]	5-45				
0027x	Keypoint		5-44				

[1] Not implemented on IO
[2] Implemented on IO only

Code	Description	Page	Code	Description	Page
0030d	Load (d)	5-9	1030d	Load (d) long	5-9
0031d	Add (d)	5-11	1031d	Add (d) long	5-11
0032d	Subtract (d)	5-11	1032d	Subtract (d) long	5-11
0033d	Logical difference (d)	5-16	1033d	Logical difference (d) long	5-16
0034d	Store (d)	5-9	1034d	Store (d) long	5-9
0035d	Replace add (d)	5-17	1035d	Replace add (d) long	5-17
0036d	Replace add one (d)	5-17	1036d	Replace add one (d) long	5-17
0037d	Replace subtract one (d)	5-17	1037d	Replace subtract one (d) long	5-18
0040d	Load ((d))	5-10	1040d	Load ((d)) long	5-10
0041d	Add ((d))	5-12	1041d	Add ((d)) long	5-12
0042d	Subtract ((d))	5-12	1042d	Subtract ((d)) long	5-12
0043d	Logical difference A and ((d))	5-16	1043d	Logical difference A and ((d)) long	5-16
0044d	Store ((d))	5-10	1044d	Store ((d)) long	5-10
0045d	Replace add ((d))	5-18	1045d	Replace add ((d))long	5-18
0046d	Replace add one ((d))	5-18	1046d	Replace add one ((d)) long	5-18
0047d	Replace subtract one ((d))	5-19	1047d	Replace subtract one ((d)) long	5-19
0050dm	Load (m+(d))	5-10	1050dm	Load (m+(d)) long	5-10
0051dm	Add (m+(d))	5-12	1051dm	Add (m+(d)) long	5-12
0052dm	Subtract (m+(d))	5-13	1052dm	Subtract (m+(d)) long	5-13
0053dm	Logical difference (m+(d)	5-16	1053dm	Logical difference (m+(d)) long	5-16
0054dm	Store (m+(d))	5-10	1054dm	Store (m+(d)) long	5-10
0055dm	Replace add (m+(d))	5-19	1055dm	Replace add (m+(d)) long	5-19
0056dm	Replace add one (m+(d))	5-20	1056dm	Replace add one (m+(d)) long	5-20
0057dm	Replace subtract one (m+(d))	5-20	1057dm	Replace subtract one (m+(d)) long	5-20
0060d	Central read from (A) to d [1]	5-29	1060d	Central read from (A) to d long	5-30
0061dm	Central read (d) words from (A) to m [1]	5-31	1061dm	Central read (d) words from (A) to m long	5-31
0062d	Central write to (A) from d [1]	5-32	1062d	Central write to (A) from d long	5-33
0063dm	Central write (d) words to (A) from m [1]	5-34	1063dm	Central write (d) words to (A) from m long	5-34
00640cm	Jump to m if channel c active	5-36	1064Xcm	Jump to m if channel c flag set	5-36
00641cm	Test and set channel c flag	5-36			

[1] Not implemented on IO

Code	Description	Page	Code	Description	Page
00650cm	Jump to m if channel c inactive	5-36	1065Xcm	Jump to m if channel c flag clear	5-36
00651cm	Clear channel c flag	5-36			
00660cm	Jump to m if channel c full	5-36	1066m	Jump to m on uncorrected error [2]	5-37
00661cm	Test and clear channel c error flag set	5-36			
00670cm	Jump to m if channel c empty	5-37			
00671cm	Test and clear channel c error flag clear	5-37			
00700c	Input to A from channel c when active and full	5-37	1070cm	Input (m) channel words to cent. mem. (A) long [2]	5-48
00701c	Input to A from channel c if active	5-37			
0071Xcm	Input A words to m from channel c	5-39	1071Xcm	Input A words to m from channel c packed [1]	5-40
00720c	Output from A on channel c when active	5-41	1072cm	Output (m) channel words from cent.mem. (A) long[2]	5-41
00721c	Output from A on channel c if active	5-41			
0073Xcm	Output A words from m on channel c	5-42	1073Xcm	Output A words from m on channel c packed [1]	5-42
00740c	Activate channel c when inactive	5-42	1074c	Master clear channel c [2]	5-42
00741c	Unconditionally activate channel c	5-42			
00750c	Deactivate channel c	5-43	1075	Pass	5-44
00751c	Unconditionally deactivate channel c	5-43			
00760c	Function A on channel c when inactive	5-43	1076	Pass	5-44
00761c	Function A on channel c if inactive	5-43			
00770cm	Function m on channel c when inactive	5-43	1077	Pass	5-44
00771cm	Function m on channel c if inactive	5-43			

[1] Not implemented on IO
[2] Implemented on IO only

APPENDIX F: PP Instruction Address Modes

Instruction Type	Addressing Mode				
	No Address	Constant	Direct	Indirect	Memory
Load	0014	0020	0030,1030	0040,1040	0050,1050
Add	0016	0021	0031,1031	0041,1041	0051,1051
Subtract	0017	-	0032,1032	0042,1042	0052,1052
Logical Difference	0011	0023	0033,1033	0043,1043	0053,1053
Store	-	-	0034,1034	0044,1044	0054,1054
Replace Add	-	-	0035,1035	0045,1045	0055,1055
Replace Add One	-	-	0036,1036	0046,1046	0056,1056
Replace Subtract One	-	-	0037,1037	0047,1047	0057,1057
Long Jump	-	-	-	-	0001
Return Jump	-	-	-	-	0002
Unconditional Jump	0003	-	-	-	-
Zero Jump	0004	-	-	-	-
Nonzero Jump	0005	-	-	-	-
Positive Jump	0006	-	-	-	-
Negative Jump	0007	-	-	-	-
Shift	0010	-	1010 [2]	-	-
Logical Product	0012	0022	1022	1023	1024
Selective Clear	0013	-	-	-	-
Load Complement	0015	-	-	-	-

[2] Implemented on IO only

APPENDIX G

Bit 0 - Data Read

Op	Ref	Instruction	Debug When
DS	001	Load Bytes, Immediate	LO ≤ Aj+Xi+D ≤ HI
A2	005	Load Word, Indexed	LO ≤ Aj+8*Xi+8*D ≤ HI
82	006	Load Word	LO ≤ Aj+8*Q ≤ HI
A4	009	Load Bytes	LO ≤ Aj+Xi+D ≤ HI
86	013	Load Bytes, Relative	LO ≤ P+Q ≤ HI
88	014	Load Bit	LO ≤ Aj+Q+X0/8 ≤ HI
A0	016	Load Address, Indexed	LO ≤ Aj+Xi+D ≤ HI
84	017	Load Adress	LO ≤ Aj+Q ≤ HI
80	020	Load Multiple	LO ≤ Aj+8*Q ≤ HI
70	074	Decimal Sum	LO ≤ Aj+01 Ak+02 ≤ HI
71	075	Decimal Difference	
72	076	Decimal Product	
73	077	Decimal Quotient	
E4	078	Decimal Scale	LO ≤ Aj+01 ≤ HI
E5	079	Decimal Scale, Rounded	
74	083	Decimal Compare	LO ≤ Aj+01 Ak+02 ≤ HI
77	084	Byte Compare	
E9	085	Byte Compare, Collated	LO ≤ Aj+01 Ak+02 Ai+D ≤ HI
F3	086	Byte Scan While Nonmember	LO ≤ Ak+01 Ai+D ≤ HI
EB	088	Byte Translate	LO ≤ Aj+01 Ai+D ≤ HI
76	089	Move Bytes	LO ≤ Aj+01 ≤ HI
ED	091	Edit	LO ≤ Aj+01 Ai+D ≤ HI
75	092	Numeric Move	LO ≤ Aj+01 ≤ HI
B5	115	Call Indirect	LO ≤ Aj+8*Q ≤ HI
04	117	Return	LO ≤ A2 ≤ HI
06	118	Pop	

Op	Ref	Instruction	Debug When
14	124	Test and Set Bit	LO ≤ Aj+X0/8 ≤ HI
B4	125	Compare Swap	LO ≤ Aj ≤ HI
FA	155	Compare Immediate Data	LO ≤ Ak+01 ≤ HI
FB	156	Add Immediate Data	
44	172	Integer Vector Sum	LO ≤ Ai, Aj* ≤ HI
45	173	Integer Vector Difference	
50	176	Integer Vector Compare EQ	
51	177	Integer Vector Compare LT	
52	178	Integer Vector Compare GE	
53	179	Integer Vector Compare NE	
4D	180	Shift Vector Circular	
48	181	Logical Vector Sum	
49	182	Logical Vector Difference	
4A	183	Logical Vector Product	
4B	184	Convert Vector Integer to FP	LO ≤ Aj* ≤ HI
4C	185	Convert Vector FP to Integer	
40	186	FP Vector Sum	LO ≤ Ai, Aj* ≤ HI
41	187	FP Vector Difference	
42	188	FP Vector Product	
43	189	FP Vector Quotient	
57	190	FP Vector Summation	LO ≤ Ai ≤ HI
54	191	Merge Vector	LO ≤ Ai, Aj* ≤ HI
55	192	Gather Vector	LO ≤ Aj* ≤ HI
56	193	Scatter Vector	
58	195	FP Vector Triad, * +	LO ≤ Ai, Aj* ≤ HI
59	196	FP Vector Triad, * -	
5A	197	FP Vector Triad, + *	
5B	198	FP Vector Triad, - *	
5C	199	FP Vector Dot Product	
5D	200	Gather Vector, Index	
5E	201	Scatter Vector, Index	

* Aj is not used as an operand for Debug when broadcast is selected (see 2.12.1.3).

Bit 1 - Data Write

Op	Ref	Instruction	Debug When	
DS	003	Store Bytes, Immediate	LO ≤ Aj+Xi+D	≤ HI
A3	007	Store Word, Indexed	LO ≤ Aj+8*Xi+8*D	≤ HI
83	008	Store Word	LO ≤ Aj+8*Q	≤ HI
A5	011	Store Bytes	LO ≤ Aj+Xi+D	≤ HI
89	015	Store Bit	LO ≤ Aj+Q+X0/8	≤ HI
A1	018	Store Address, Indexed	LO ≤ Aj+Xi+D	≤ HI
85	019	Store Adress	LO ≤ Aj+Q	≤ HI
81	021	Store Multiple	LO ≤ Aj+8*Q	≤ HI
70	074	Decimal Sum	LO ≤ Ak+02	≤ HI
71	075	Decimal Difference		
72	076	Decimal Product		
73	077	Decimal Quotient		
E4	078	Decimal Scale		
E5	079	Decimal Scale, Rounded		
EB	088	Byte Translate		
76	089	Byte Move		
ED	091	Edit		
75	092	Numeric Move		
B5	115	Call Indirect	LO ≤ A0+7,mod 8	≤ HI
B0	116	Call Relative		
14	124	Test and Set Bit	LO ≤ Aj+X0/8	≤ HI
B4	125	Compare Swap	LO ≤ Aj	≤ HI
F9	154	Move Immediate Data	LO ≤ Ak+01	≤ HI
FB	156	Add Immediate Data		

Op	Ref	Instruction	Debug When
44	172	Integer Vector Sum	
45	173	Integer Vector Difference	
50	176	Integer Vector Compare EQ	
51	177	Integer Vector Compare LT	
52	178	Integer Vector Compare GE	
53	179	Integer Vector Compare NE	
4D	180	Shift Vector Circular	
48	181	Logical Vector Sum	
49	182	Logical Vector Difference	LO ≤ Ak ≤ HI
4A	183	Logical Vector Product	
4B	184	Convert Vector Int. to FP	
4C	185	Convert Vector FP to Int.	
40	186	FP Vector Sum	
41	187	FP Vector Difference	
42	188	FP Vector Product	
43	189	FP Vector Quotient	
54	191	Merge Vector	
55	192	Gather Vector	
56	193	Scatter Vector	
58	195	FP Vector Triad, * +	
59	196	FP Vector Triad, * -	
5A	197	FP Vector Triad, + *	
5B	198	FP Vector Triad, - *	
5D	200	Gather Vector, Index	
5E	201	Scatter Vector, Index	

Bit 2 - Instruction Fetch

All instructions are eligible for a debug trap on this condition, providing they fall within an address bracket defined in the debug list.

- The Load Bytes, Relative (Op.86) reference to P+Q shall not be detected.
- Unimplemented Instruction, Program Error, and Execute Algorithm shall not necessarily be detected.
- The descriptors for BDP instructions shall not be detected.
- This test is applied for each instruction rather than for each instruction word.

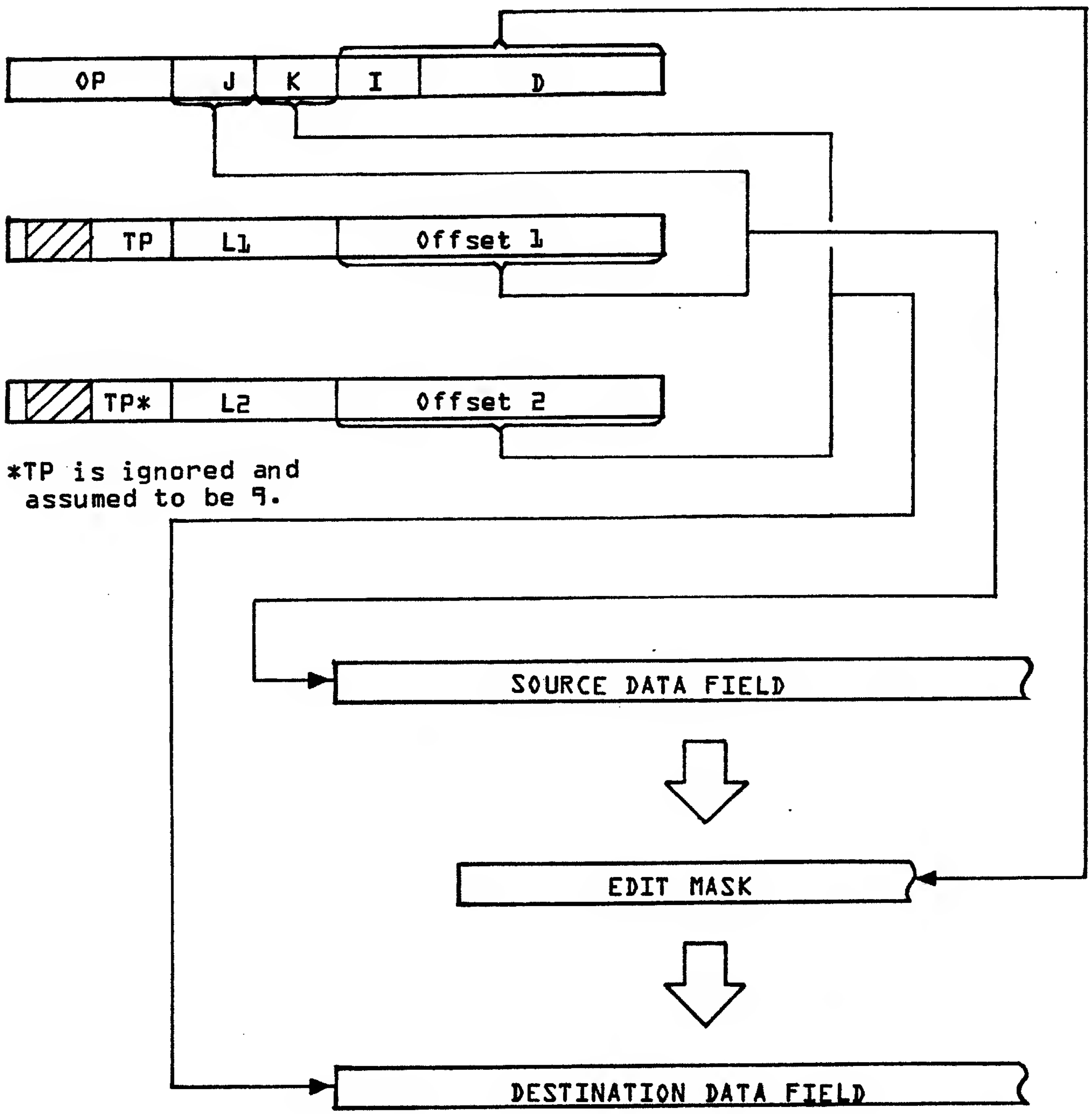
Bit 3 - Branching Instruction

Op	Ref	Instruction	Debug When
94	037	Branch on Equal	LO ≤ P+2*Q ≤ HI
95	038	Branch on Not Equal	
96	039	Branch on Greater Than	
97	040	Branch on Greater Than or Equal	
90	041	Branch on Half Word EQ	
91	042	Branch on Half Word NE	
92	043	Branch on Half Word GT	
93	044	Branch on Half Word GE	
9C	045	Branch and Increment	
9D	046	Branch on Segments Unequal	
2E	047	Branch Relitive	LO ≤ P+2*Xk ≤ HI
2F	048	Branch Inter-segment	LO ≤ Aj+2*Xk ≤ HI
98	109	FP Branch on EQ	LO ≤ P+2*Q ≤ HI
99	110	FP Branch on NE	
9A	111	FP Branch on GT	
9B	112	FP Branch on GE	
9E	113	FP Branch on Exception	
9F	134	Branch on Condition Register (see section 1.6)	
04	117	Return	LO ≤ FINAL P ≤ HI

Bit 4 - CALL Instruction

Op	Ref	Instruction	Debug When
B5	115	Call Indirect	LO ≤ CBP ≤ HI
B0	116	Call Relative	LO ≤ P+8*Q,mod 8 ≤ HI

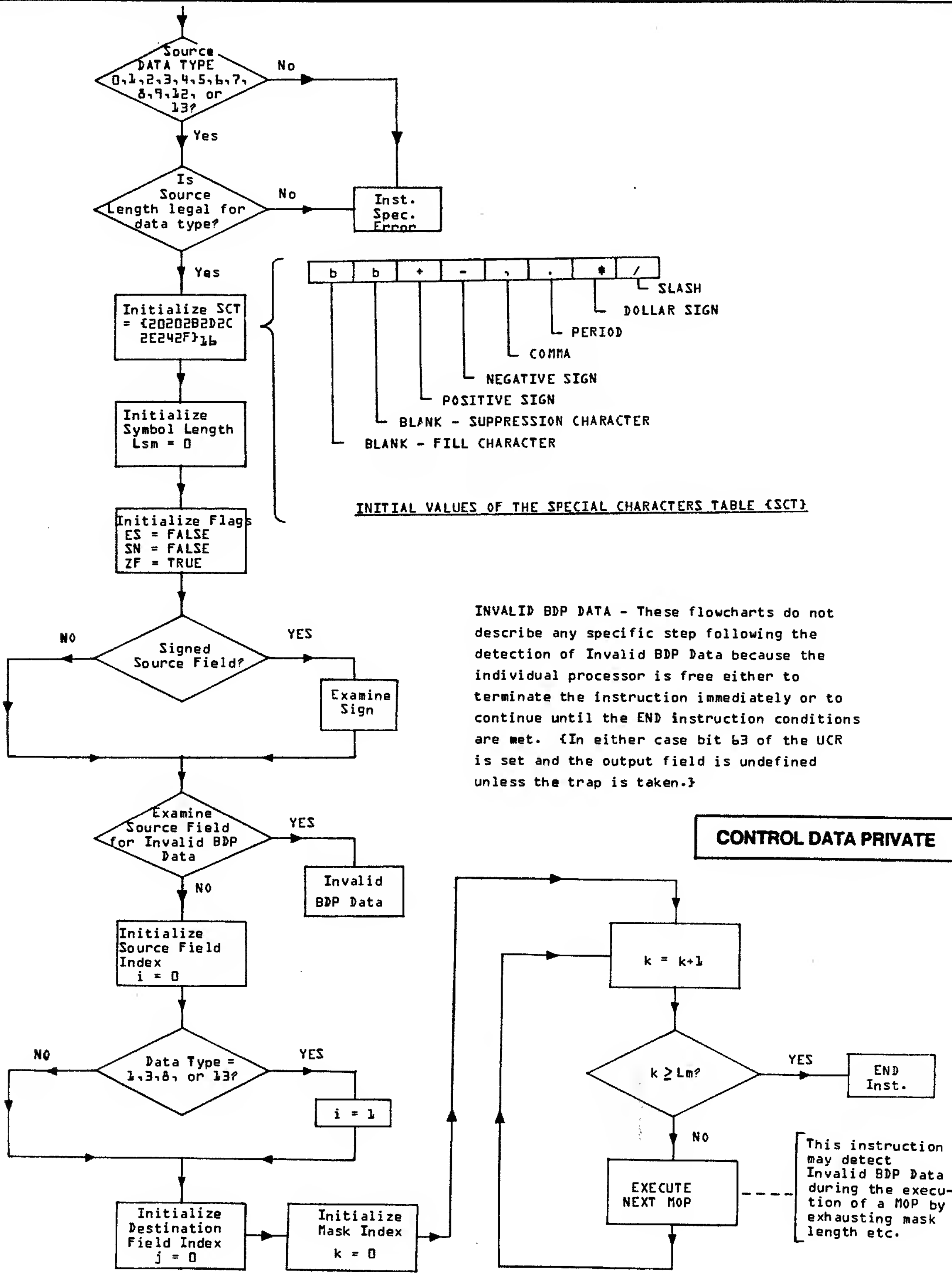
APPENDIX H: Edit Flowcharts



EDIT INSTRUCTION

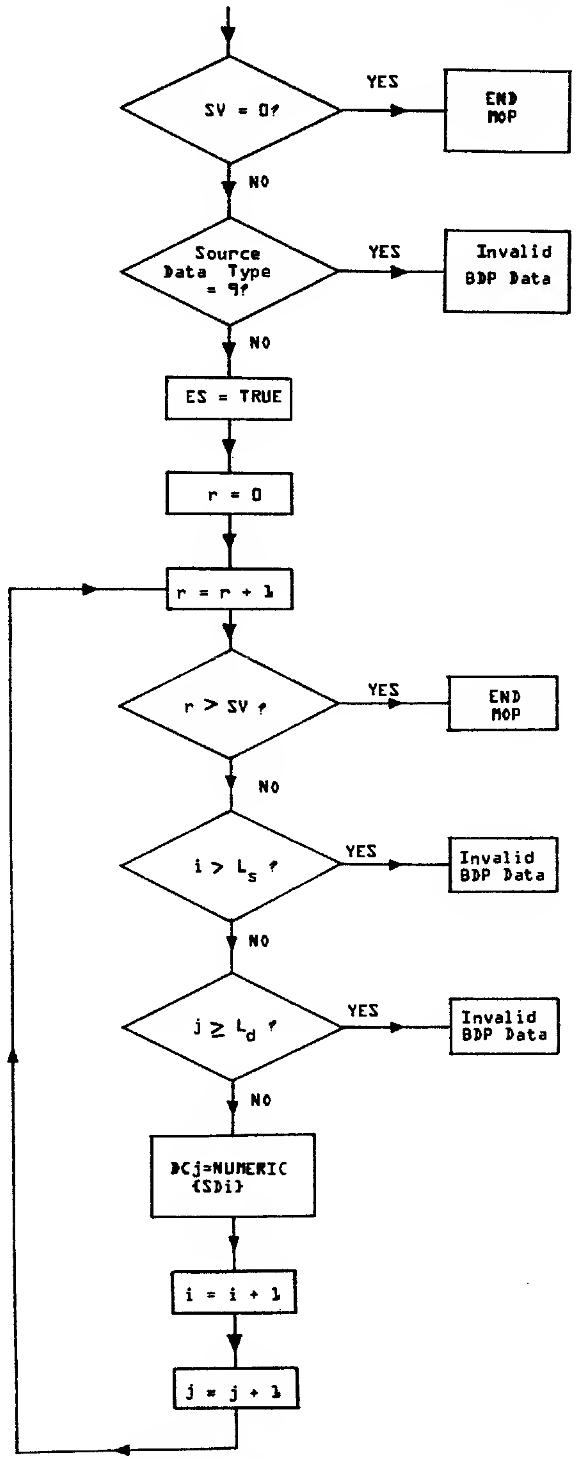
Key to Symbols Used in the Following Flowcharts

i	Index for the source field in bytes for data type 9 and in digits for all other data types (skipping slack digits on data type 1, 3 and 13 and skipping separate sign on data types 2, 3, 6, 8 and 12).
j	Index for destination field, initialized to 0.
k	Index for mask, initialized to 0.
SCi	The source character addressed by base of source field indexed by i.
SDi	The source digit addressed by base of source field indexed by i.
DCj	The destination byte addressed by base of destination field indexed by j.
MCk	The mask byte addressed by base of mask field indexed by k.
ES	End supression toggle (initialized False and then set True when end supression occurs).
ZF	Zero field toggle (initialized True and then set False when nonzero source digit is processed).
SN	Sign toggle (initialized False and then set True if source field is negative).
SCT	Special character table.
SCT_n	The (n + 1)th entry in the SCT (n must be 0-7).
SV	Specification value.
SM	The symbol.
SM_c	The cth symbol character.
L_s	Length of source field in digits (or in characters for type 9)
L_m	Length of edit mask in bytes.
L_d	Length of destination of field in bytes.
L_{sm}	Length of the symbol in bytes, initialized to 0.
r	A loop counting mechanism associated with SV.
t	A loop counting mechanism associated with L_{sm}.



INVALID BDP DATA - These flowcharts do not describe any specific step following the detection of Invalid BDP Data because the individual processor is free either to terminate the instruction immediately or to continue until the END instruction conditions are met. (In either case bit 63 of the UCR is set and the output field is undefined unless the trap is taken.)

EDIT OVERVIEW, INCLUDING INITIALIZATION



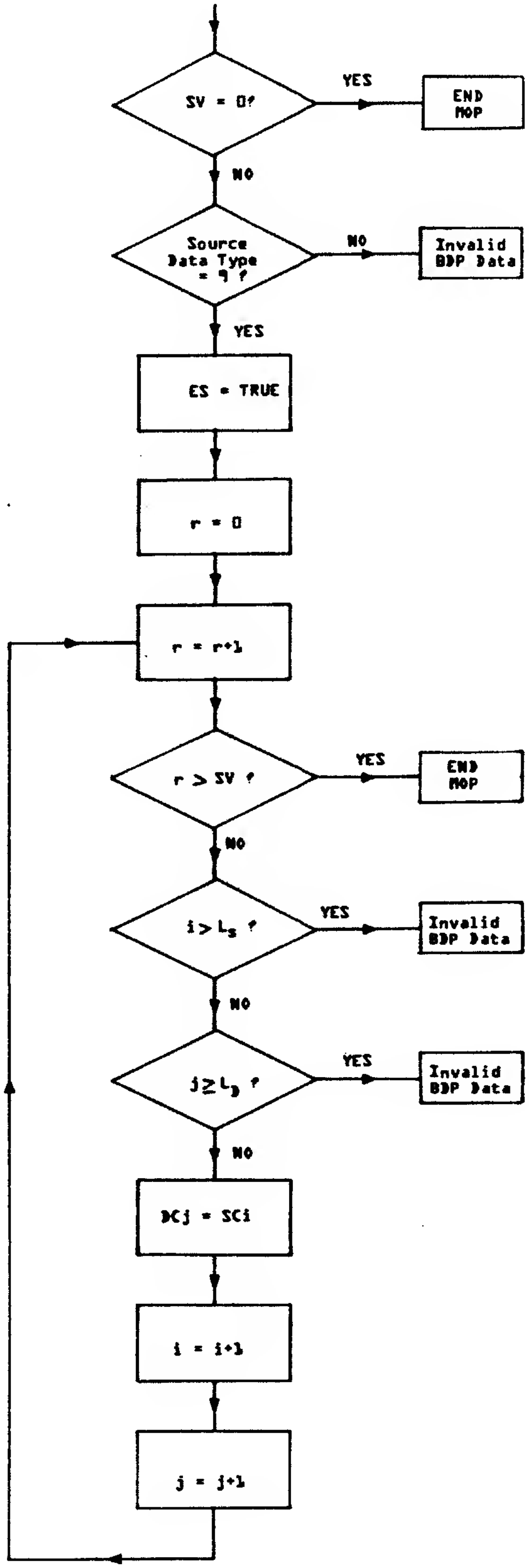
MOP D - MOVE SOURCE DIGITS

Notes:

1. This MOP translates 1-15 digits in the source field to their equivalent ASCII characters and copies them to the destination field.

2. The function NUMERIC is flow-charted elsewhere, and insures that the data being translated is valid, numeric data.

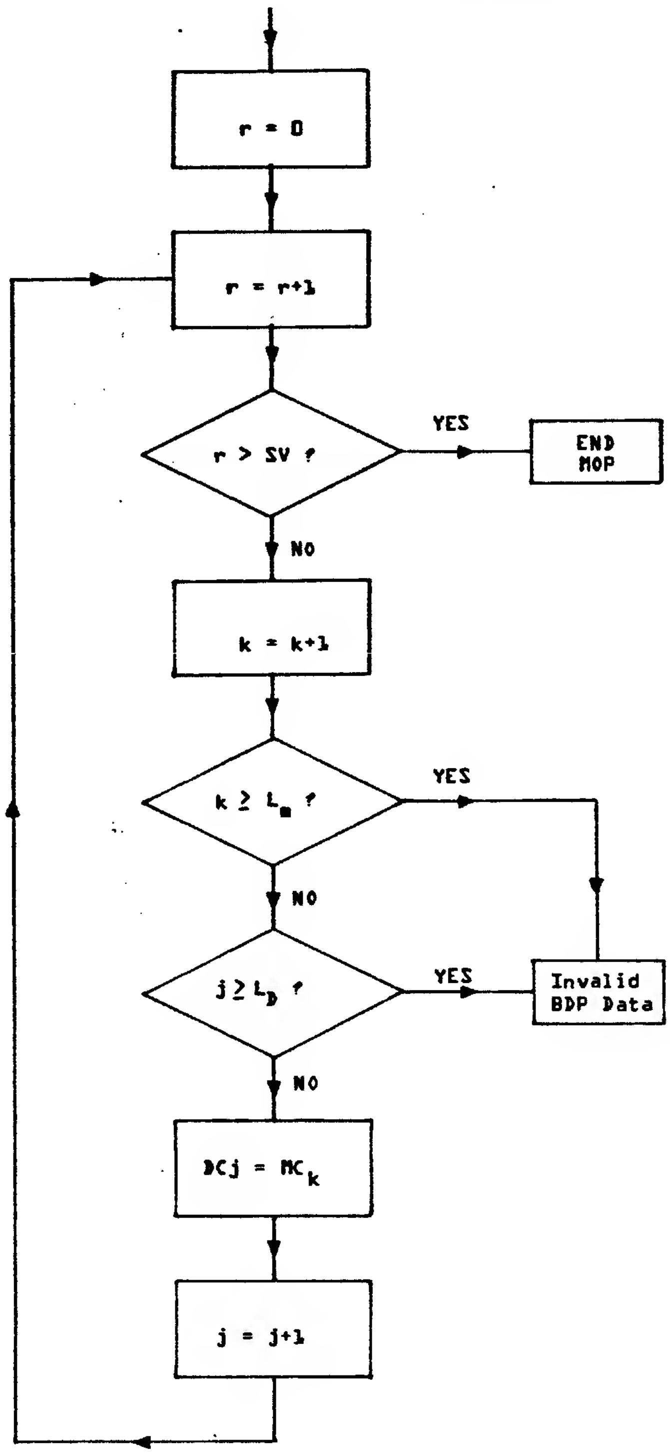
3. Set ES true if SV ≠ 0.



Notes:

1. This MOP copies 1-15 characters froe the source field to the destination field.

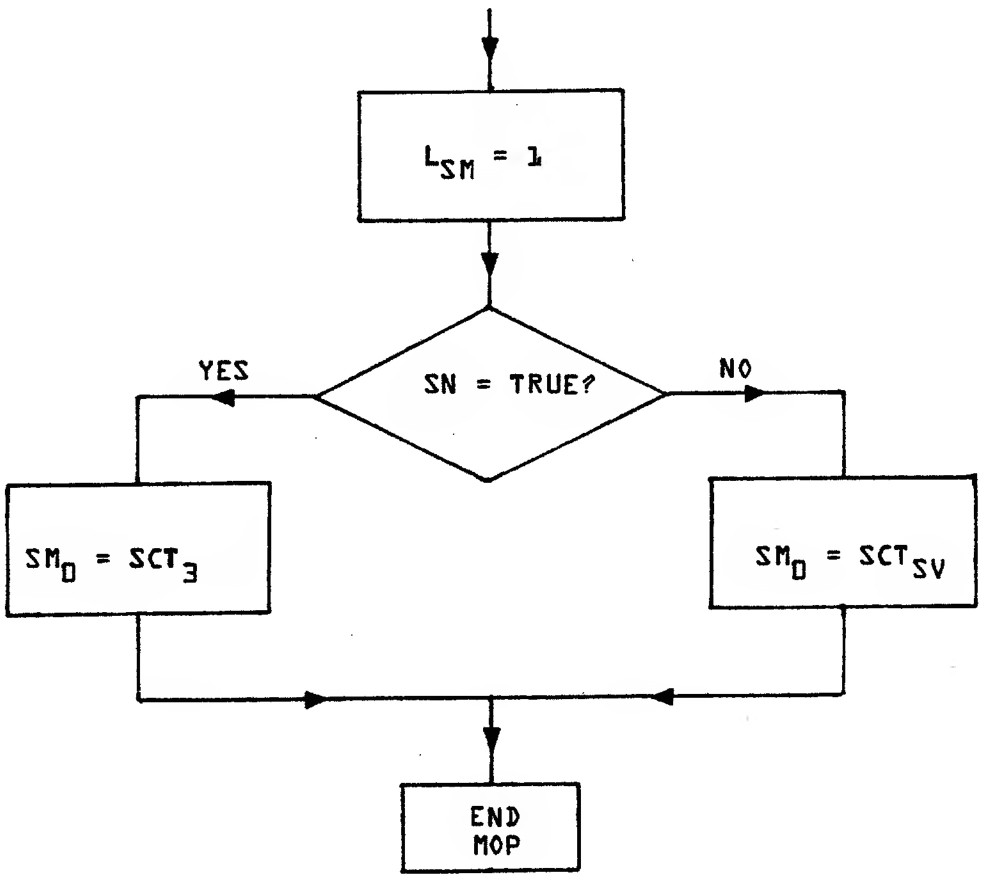
MOP 1 - MOVE SOURCE CHARACTERS



MOP 4 - MOVE MASK CHARACTERS

Notes:

1. This MOP moves 1-15 characters from the edit mask (in essence a micro-op string) to the destination field.
2. Any of the source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



MOP5 - SELECT SIGN AS SYMBOL

Notes:

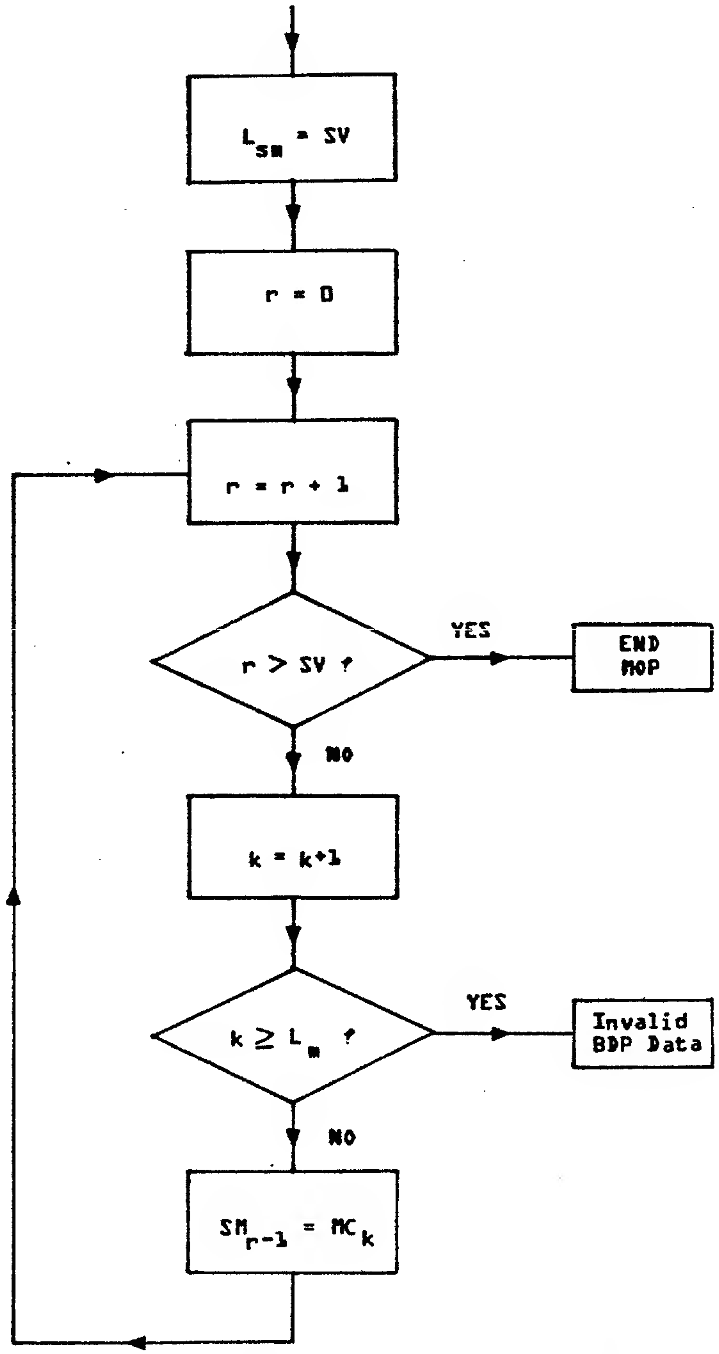
1. This MOP sets the symbol to a single character representing the sign of the source data field.

2. If the source data field is negative, then the sign is either set to minus (default value in the SCT) or to the value which has been stored in SCT(3).

3. If the source data field is positive, then the sign is set to a value selected from the SCT indexed by the least significant three bits of the specification value. Assuming a default SCT SV would normally have a value equal to 1 or 2 corresponding to blank and plus.

4. All values of SV are legal although only the rightmost three bits are interpreted when SV is used as an index.

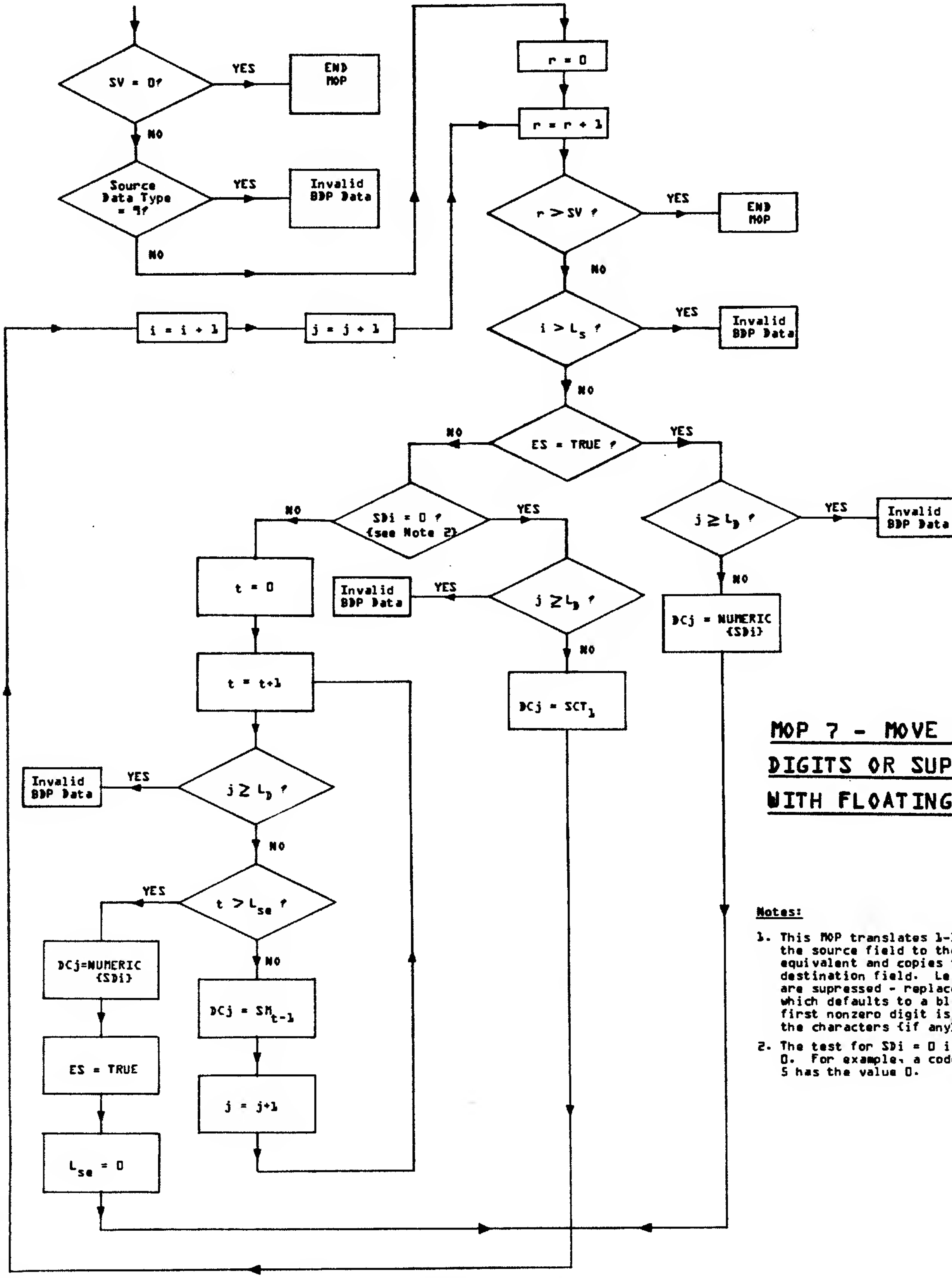
5. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



MOP 6 - SELECT MASK CHARACTERS AS SYMBOL

Notes:

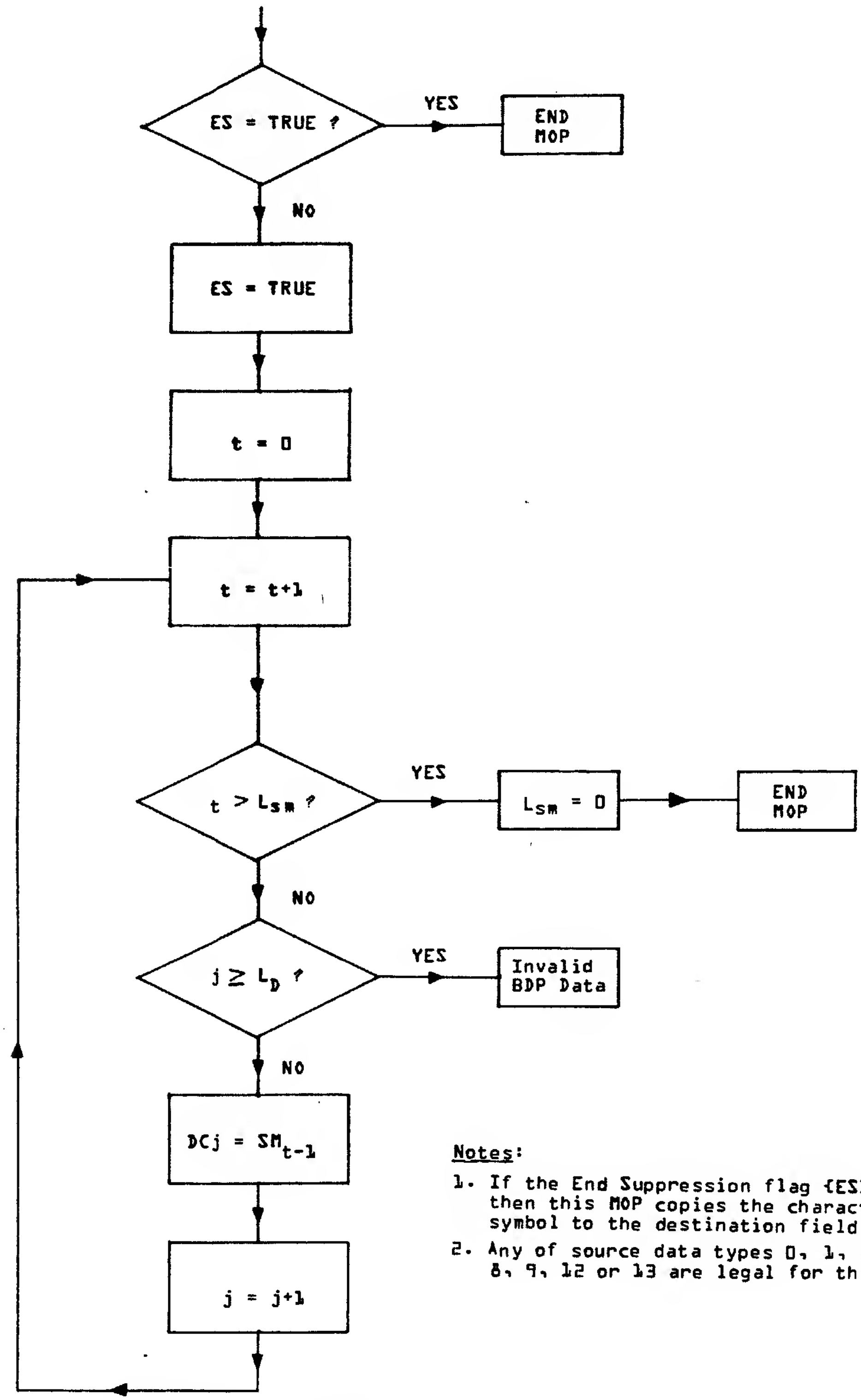
1. This MOP copies 1-15 characters from the edit mask to the symbol.
2. All values of SV are legal for this MOP.
3. Any of the source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



MOP 7 - MOVE SOURCE DIGITS OR SUPRESS WITH FLOATING SYMBOL

Notes:

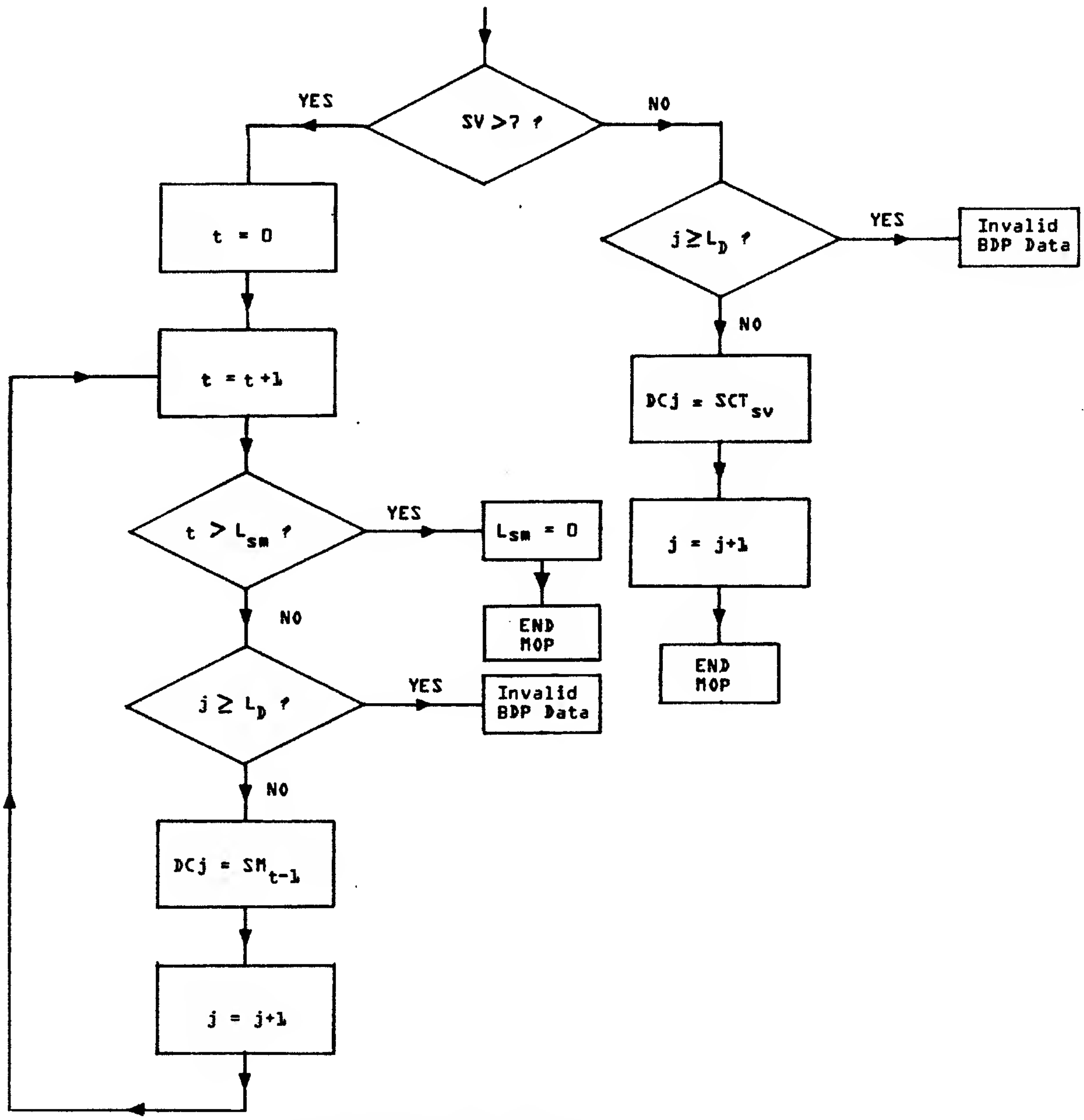
1. This MOP translates 1-15 digits from the source field to their ASCII equivalent and copies them to the destination field. Leading zeros are supressed - replaced by SCT(1) which defaults to a blank - and the first nonzero digit is preceded by the characters (if any) in the symbol.
2. The test for SDi = 0 is for the value 0. For example, a code of 3C on type 5 has the value 0.



MOP 8 - END FLOAT

Notes:

1. If the End Suppression flag (ES) is not set, then this MOP copies the characters in the symbol to the destination field.
2. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



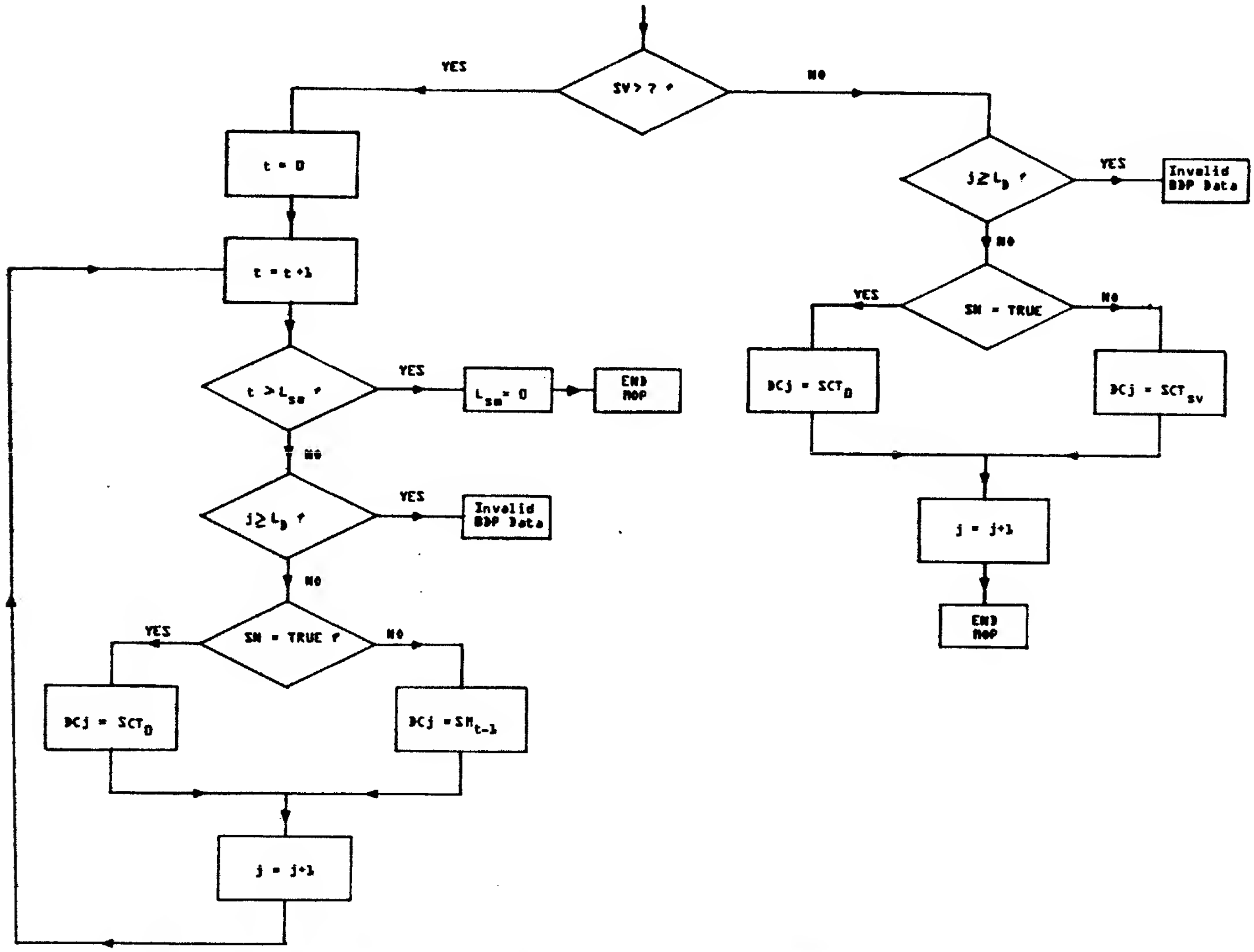
MOP 9 - INSERT SYMBOL OR SCT CHARACTER

Notes:

1. This MOP either inserts the symbol characters or a character from the SCT into the destination field.

2. This MOP is controlled by the SV field. The most significant bit of this field is used as a flag. If set, then the symbol is inserted into the destination field, otherwise the SVth character from the SCT is inserted into the destination field.

3. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



MOP A - INSERT SYMBOL OR SCT CHARACTER IF SOURCE IS POSITIVE, ELSE INSERT BLANKS

Notes:

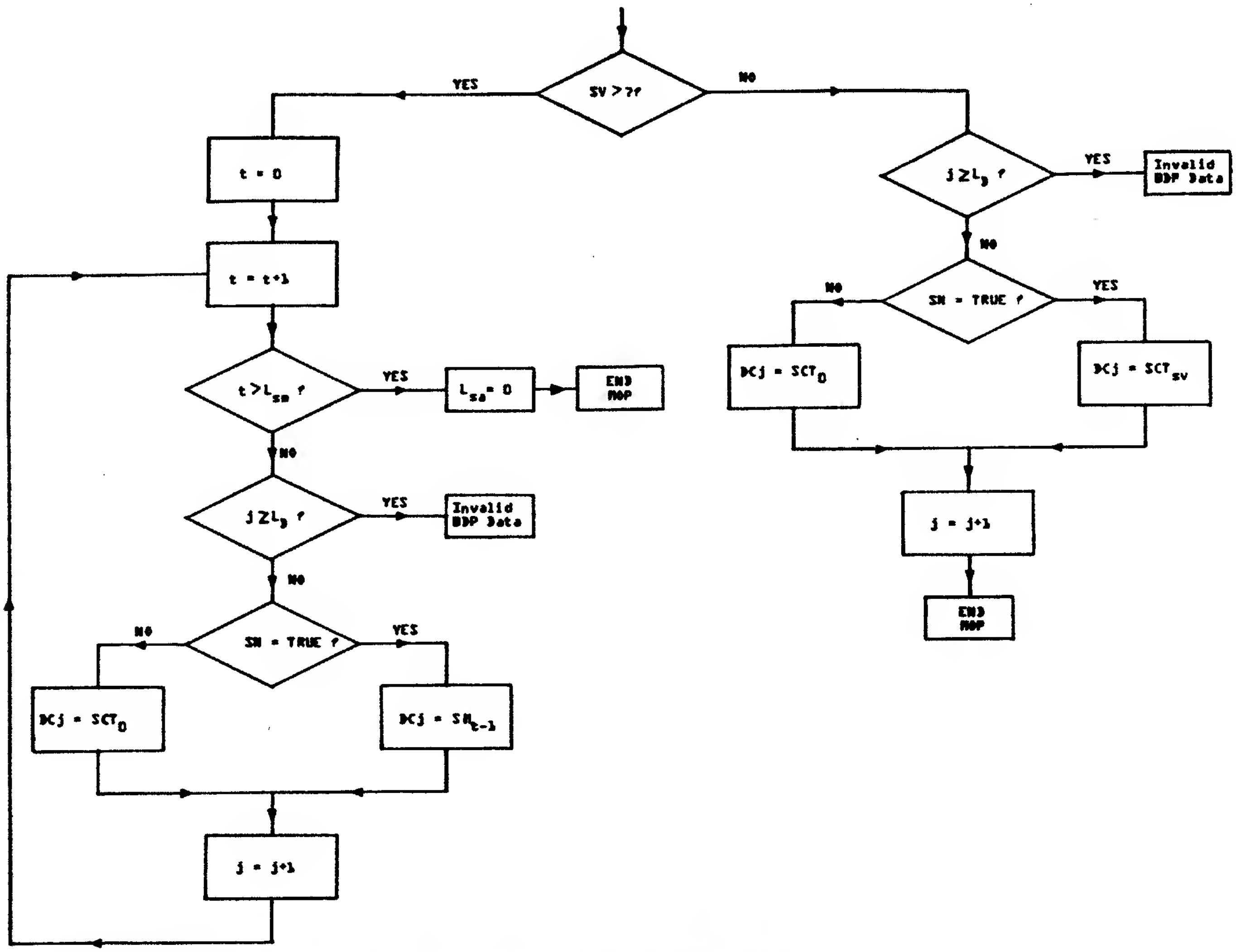
1. SV > 7

   Copy the Symbol to the destination field when the source field is positive, otherwise copy SCT_0 once for each character in the Symbol.

2. SV ≤ 7

   Copy SCT_{SV} once to the destination field when the source field is positive, otherwise copy SCT_0 once.

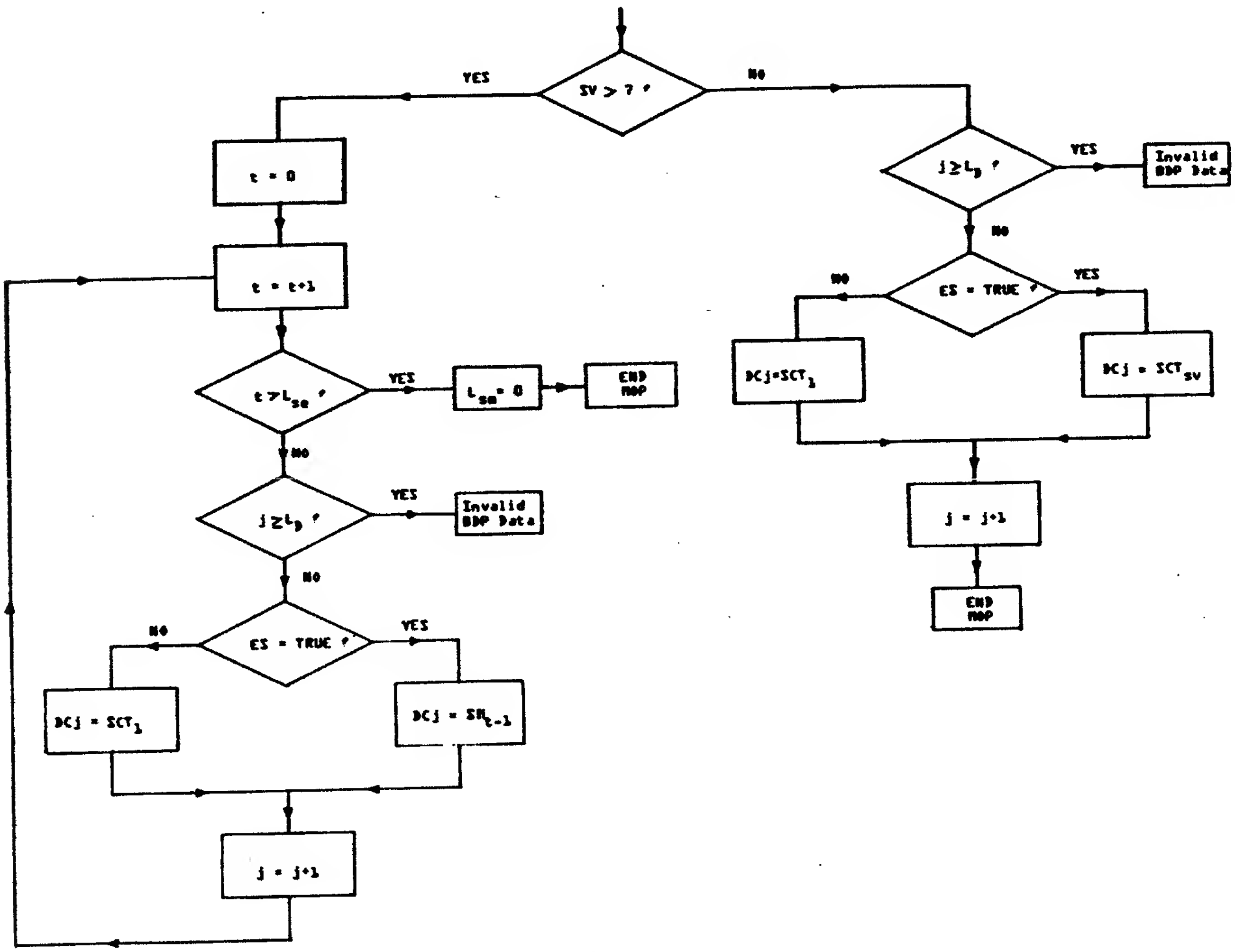
3. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



MOP B - <u>INSERT SYMBOL OR SCT CHARACTER IF SOURCE IS NEGATIVE, ELSE INSERT BLANKS</u>

Notes: 1. This MOP is identical in all respects to MOP A except that the blank insertion occurs for a <u>positive</u> rather than negative source field.

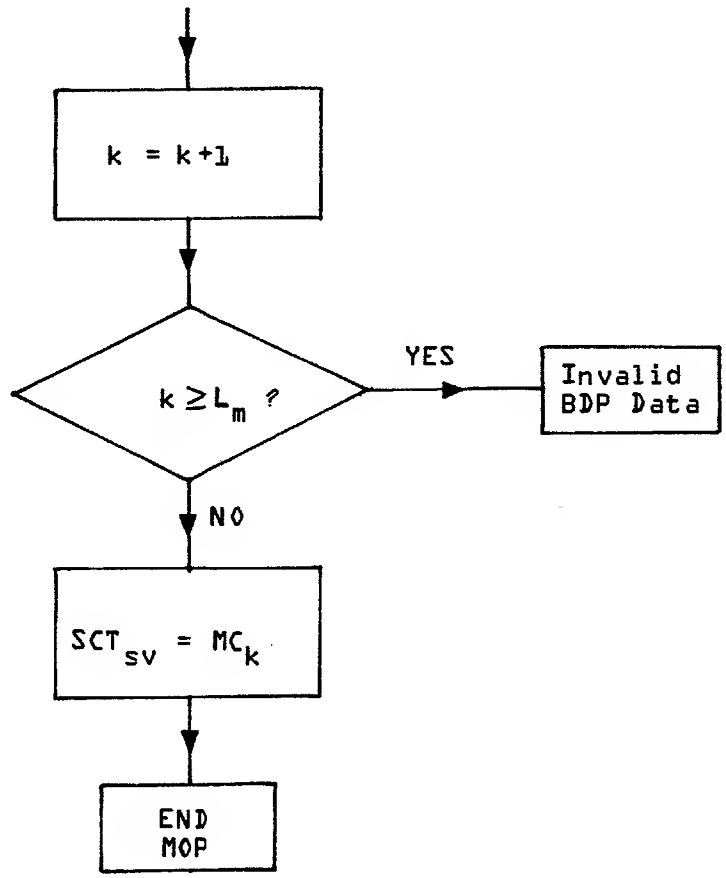
2. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



<u>MOP C - INSERT SYMBOL OR SCT CHARACTER, UNLESS SUPPRESSION</u>

Notes: 1. This MOP is identical in all respects to MOP A except that the blank insertion occurs only when zero suppression is in effect.

2. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, or 13 are legal for this MOP.

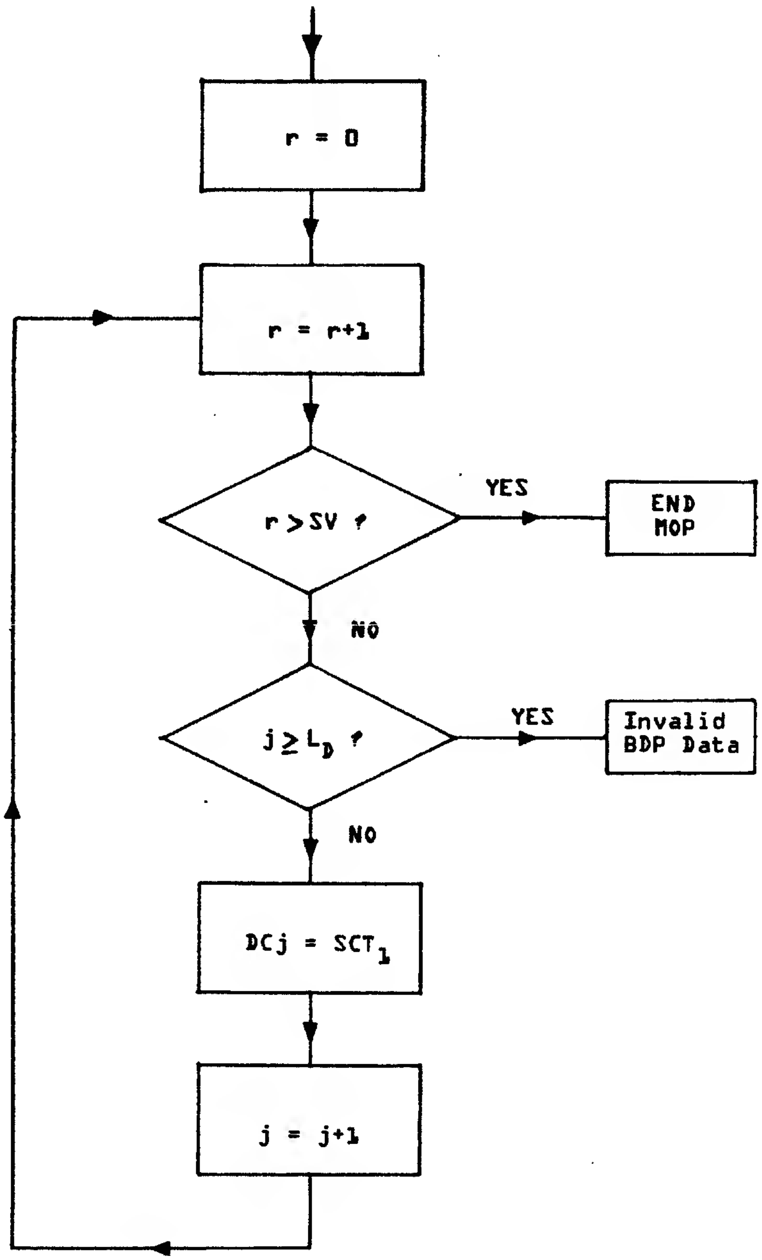


MOP D - WRITE SCT ENTRY

Notes: 1. This MOP copies the next character from the edit mask into the SVth character of the SCT.

2. Only the low-order three bits of the SV are used by this MOP.

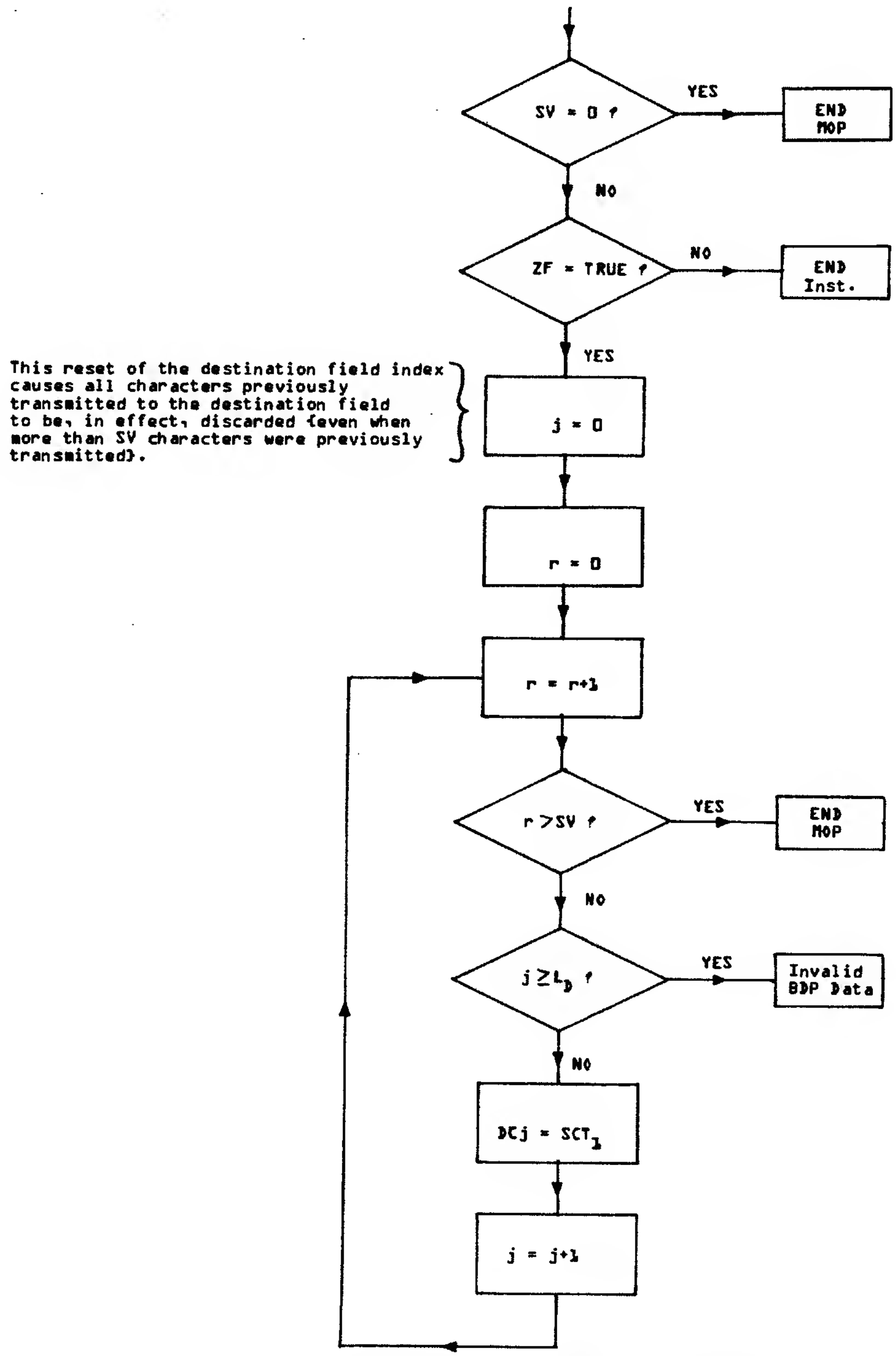
3. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.



MOP E - SPREAD SUPPRESSION CHARACTER

Notes: 1. This MOP copies the suppression character (from SCT(1)) into the destination field SV times.

2. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, or 13 are legal for this MOP.

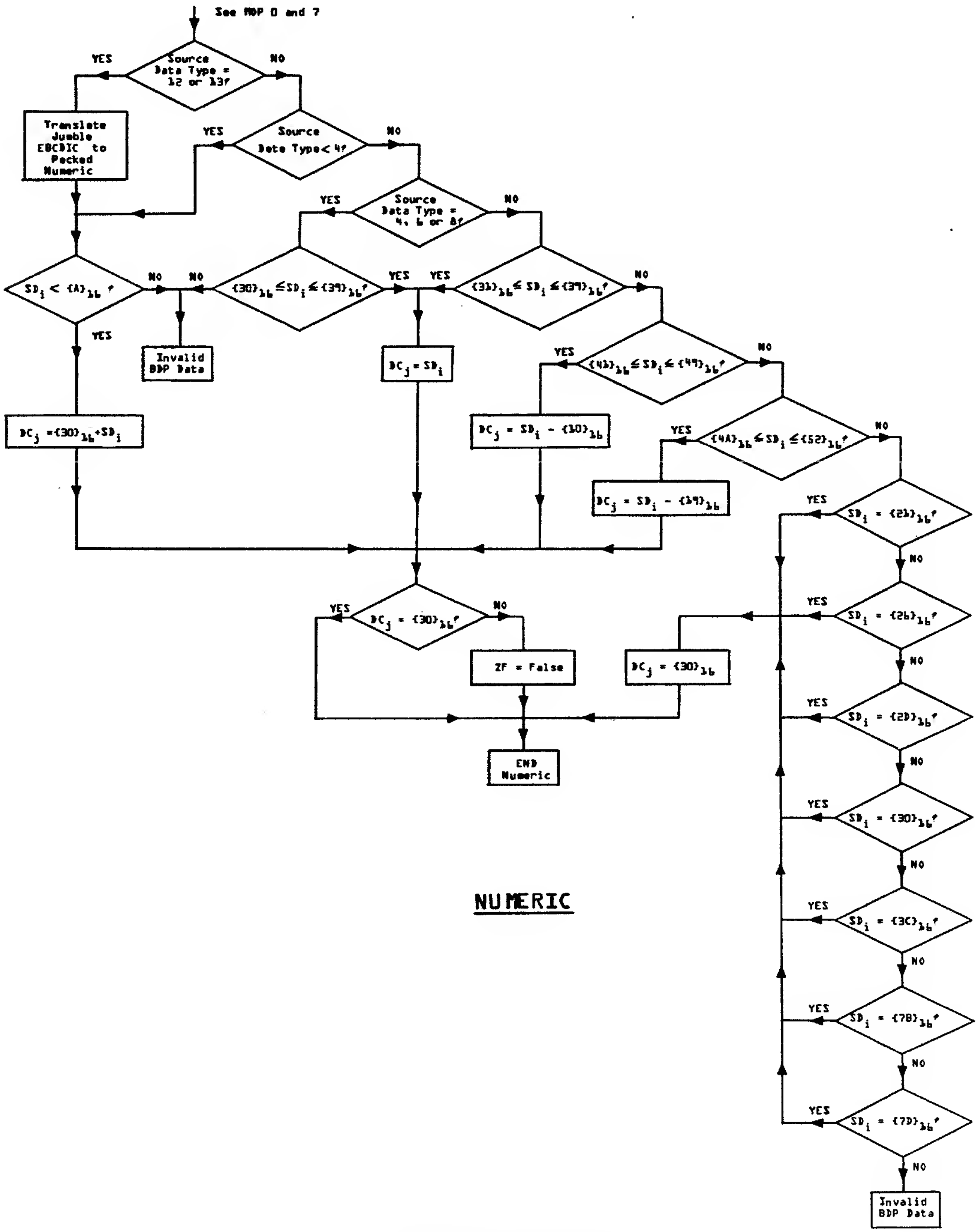


MOP F - RESET AND SUPRESS ON ZERO FIELD

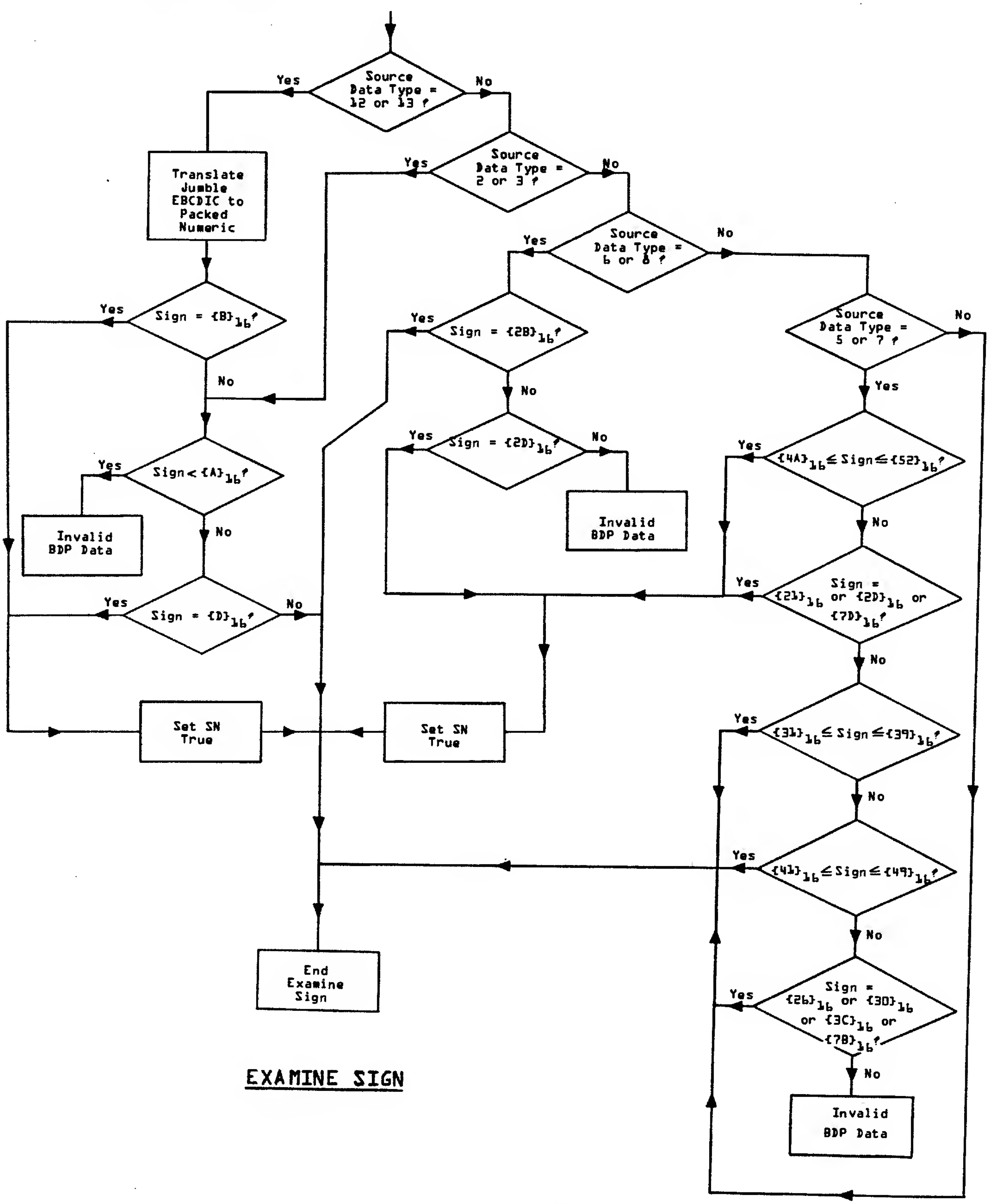
Notes: 1. This MOP functions only for source fields with a zero value. A non-zero source field when SV≠0 causes the termination of the edit instruction (not just this MOP).

2. For a zero source field, SV suppression characters are copied into the destination field.

3. Any of source data types 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12 or 13 are legal for this MOP.


See MOP D and 7
Source Data Type = 12 or 13?
YES
NO
Translate Jumble EBCDIC to Packed Numeric
Source Data Type< 4?
Source Data Type = 4, 6 or 8?
SDi < {A}16 ?
{30}16 ≤ SDi ≤ {39}16?
{31}16 ≤ SDi ≤ {39}16?
Invalid BDP Data
DCj = SDi
DCj = {30}16 + SDi
{41}16 ≤ SDi ≤ {49}16?
DCj = SDi - {10}16
{4A}16 ≤ SDi ≤ {52}16?
DCj = SDi - {19}16
SDi = {2B}16?
SDi = {26}16?
SDi = {2D}16?
SDi = {30}16?
SDi = {3C}16?
SDi = {7B}16?
SDi = {7D}16?
DCj = {30}16?
ZF = False
DCj = {30}16
END Numeric
Invalid BDP Data

<u>NUMERIC</u>



EXAMINE SIGN

APPENDIX I

Exception Conditions - UTP

This appendix lists each way that an exception condition can cause the setting of Invalid Segment/Ring Number Zero, Access Violation, Address Specification Error or Page Table Search Without Find. For each of these exceptions, it specifies the proper action relative to the contents of the Untranslatable Pointer (UTP). In addition, the appendix reflects whether the instruction will inhibit or complete execution.

The first entries describe the exception detection when sequential (non-branch) execution leads to a situation where the next sequential instruction has either an Address Specification Error or Page Table Search Without Find associated with it. Exception detection when the branch is taken (execution continues at new P rather than sequential P) is described under each instruction which generates a new P (OP 04, 2E, 2F, QO-9F, BO, B4, B5 and Trap). The action on exchange operations is described in 2.8. Also see 2.1.3.4. For branch taken, complete means that the P in the Exchange Package or Stack Frame Save Area points to the branch destination.

EXCEPTION CONDITION	Complete or Inhibit	MCR60 Invalid Segment/Ring No. Zero	MCR54 Access Violation	MCR52 Address Spec. Error	MCR57 Page Table Search wo/Find	UTP CONTENTS (48 bit PVA except where noted for Op. 05 and 17.)
Instruction fetch due to sequential execution (including normal exit on branch)						
Final P has bit 32 = 1	*			X		Final P
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	*			X		P plus some index having bit 32 = 1
Final P has Page Fault	*				X	Final P
Final P points to an instruction whose first portion is in memory but whose latter portion or BDP descriptor causes a Page Fault	*				X	P plus some index pointing to the missing instruction portion
Debug						
Debug List Pointer accesses Invalid Segment	I	X				Debug List Pointer OR Debug List Pointer plus Index
Debug List Pointer points to a nonreadable segment (RP = 00)	I		X			
Debug List Pointer.RN > SDE(Debug List Ptr).R2	I		X			
Debug List Pointer key/lock test fails	I		X			
Debug List Pointer plus Index has bit 32 = 1	I			X		
Debug List Pointer ≠ 0, mod 8	I			X		
Debug List Pointer plus Index points to entry not in memory - Page Fault	I				X	

* Execution will be completed for the instruction prior to the address which could not be fetched due to the exception.

EXCEPTION CONDITION	Complete or Inhibit	MCR60 Invalid Segment/Ring No. Zero	MCR54 Access Violation	MCR52 Address Spec. Error	MCR57 Page Table Search wo/Find	UTP CONTENTS (48 bit PVA except where noted for Op. 05 and 17.)
<u>04 Return</u>						
*Final P accesses Invalid Segment	E	X				Final P
†Initial A2 accesses Invalid Segment	I	X				Initial A2 + any index < 256
Load of an A register from SFSA with RN = 0	C	X				(Not altered)
†Initial A2 points to nonreadable segment (RP=00)	I		X			Initial A2 plus any index up to 256
†Initial A2.RN > SDE(Initial A2).R2	I		X			Initial A2 plus any index up to 256
†Initial A2 key/lock test fails	I		X			Initial A2 plus any index up to 256
*Final P points to nonexecutable segment	E		X			Final P
*Final P key/lock test fails	E		X			Final P
*Final P.RN > SDE(Final P).R2 or < SDE(Final P).R1	E		X			Final P
†Initial A2 ≠ 0, mod 8	I			X		Initial A2
†Initial A2 + any of the indices into the SFSA (up to potentially 256) has bit 32 = 1	I			X		Initial A2 plus any index up to 256 which causes bit 32=1
Final P has bit 32 = 1	E			X		Final P
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	C			X		Final P
*Final P has bit 63 = 1	E			X		Final P
†Initial A2 + any of the indices into the SFSA (up to potentially 256) has a Page Fault	I				X	Initial A2 plus any index up to 256 which causes a Page Fault
Final P has Page Fault	E				X	Final P
Final P points to an instruction whose first portion is in memory but whose latter portion or BDP descriptor causes a Page Fault	C				X	Final P
<u>05 Purge Buffer</u>						
PVA in Xj accesses Invalid Segment (k = 3-7,A,B)	I	X				PVA from Xj
PVA in Xj has bit 32 = 1 (k = 3-7,A,B)	I			X		PVA from Xj
SVA in Xj has bit 32 = 1 (k = 0,1,8,9)	I			X		SVA from Xj

* Test not required as part of RETURN operation when P comes from the SFSA Pushdown rather than from central memory.

† When this condition is encountered on a RETURN operation which has retrieved part of the SFSA from a SFSA Pushdown, an Environmental Specification Error will be reported rather than or in addition to the result shown in the table. Note that the environment is damaged and cannot be restarted.

EXCEPTION CONDITION	Complete or Inhibit	MCR60 Invalid Segment/Ring No. Zero	MCR54 Access Violation	MCR52 Address Spec. Error	MCR57 Page Table Search wo/Find	UTP CONTENTS (48 bit PVA except where noted for Op. 05 and 17.)
<u>06 Pop</u>						
*Initial A2 accesses Invalid Segment	I	X				Initial A2 plus any index up to 24
†A1 or A2 from SFSA has RN = 0	C	X				(Not altered)
*Initial A2 points to nonreadable segment (RP=00)	I		X			
*Initial A2.RN > SDE(Initial A2).R2	I		X			Initial A2 plus any index
*Initial A2 key/lock test fails	I		X			
*Initial A2 ≠ 0, mod 8	I			X		Initial A2
*Initial A2 plus 0, 8, 16 or 24 has bit 32 = 1	I			X		Initial A2 plus 0, 8, 16, 24
*Initial A2 plus 0, 8, 16 or 24 has Page Fault	I				X	PVA of any missing portion of SFSA
<u>14 Test and Set Bit</u>						
Aj + XOR/8 accesses invalid segment	I	X				
Aj + XOR/8 points to nonreadable or nonwritable segment	I		X			
(Aj + XOR/8).RN > SDE(Aj).R1	I		X			Aj + XOR/8
Aj + XOR/8 key/lock test fails	I		X			
Aj + XOR/8 has bit 32 = 1	I			X		
Aj + XOR/8 has Page Fault	I				X	
<u>16 Test and Set Page</u>						
Aj access invalid segment	I	X				Aj
Aj has bit 32 = 1	I			X		Aj
<u>17 Load Page Table Index</u>						
Xj has bit 32 = 1	I			X		SVA from Xj
<u>2E Branch Relative</u>						
Final P has bit 32 = 1	I			X		
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	C			X		
Final P has Page Fault — P3,0	C				X	Final P
Final P has Page Fault — P1,P2	I				X	
Final P points to an instruction whose first portion is in memory but whose latter portion or BDP descriptor causes a Page Fault	C				X	

* Test not required on POP when any portion of the SFSA is retrieved from the SFSA Pushdown.
† Test not required on POP when both A1 and A2 are retrieved from the SFSA Pushdown.

EXCEPTION CONDITION	Complete or Inhibit	MCR60 Invalid Segment/Ring No. Zero	MCR54 Access Violation	MCR52 Address Spec. Error	MCR57 Page Table Search wo/Find	UTP CONTENTS (48 bit PVA except where noted for Op. 05 and 17.)
<u>2F Intersegment Branch</u>						
Final P accesses Invalid Segment	I	X				Final P (final P.RN is that ring number which would be used on the next instruction fetch)
Final P accesses nonexecutable segment	I		X			
Final P.RN > SDE(P).R2 or <SDE(P).R1	I		X			
Final P has bit 32 = 1	I			X		
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	C			X		
Initial Aj ≠ 0, mod 2	I			X		Initial Aj or Aj plus index
Final P has Page Fault P3,0	C				X	Final P
P1,P2	I				X	
Final P points to an instruction whose first portion is in memory but whose latter portion or BDP descriptor causes a Page Fault	C				X	
4X,5X Vectors 70-77, EX, FX BDP 80-86, 88, 89 Load/Store <u>A0-A5, DX Load/Store</u>						
A register as loaded from memory has RN = 0 (Op. 80, 84, A0)	C	X				(Not altered)
Address accesses Invalid Segment	I	X				Address
Attempt to read a nonreadable segment	I		X			Any of the addresses generated by this instruction which point into the segment having the access violation
Attempt to write a nonwritable segment	I		X			
Attempt to read when RN > R2	I		X			
Attempt to write when RN > R1	I		X			
Key/lock test fails	I		X			
Address has bit 32 = 1	I			X		Address *
Address ≠ 0, mod 8 (Op. 4X, 5X, 80, 81, F4)	I			X		Address *
Instruction execution requires data which is not totally in memory	I				X	Any of the addresses generated by this instruction which point into the missing page.

* Any of the addresses generated by this instruction which have an Address Spec. Error.

EXCEPTION CONDITION	Complete or Inhibit	MCR60 Invalid Segment/Ring No. Zero	MCR54 Access Violation	MCR52 Address Spec. Error	MCR57 Page Table Search wo/Find	UTP CONTENTS {48 bit PVA except where noted for Op. 05 and 17.
90-9E Conditional Branch Instructions 9F Branch On Condition Register						
Final P has bit 32 = 1 (Branch taken)* P1	I			X		Final P
P2,P3,θ	C			X		
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	C			X		
Final P has Page Fault (Branch Taken)* P1	I				X	
P2,P3,θ	C				X	
Final P points to an instruction whose first portion is in memory but whose latter portion or BDP descriptor causes a Page Fault	C				X	
B0 Call Relative						
A0 accesses invalid segment	I	X				A0 or A0 Rounded Up
A0 points to nonwritable segment	I		X			A0 plus index up to SFSA length
A0.RN > SOE(A0).R1	I		X			
A0 key/lock test fails	I		X			
A0 + any of the indices into the SFSA (up to the Stack Frame length) has bit 32 = 1	I			X		A0 plus any index into SFSA which causes bit 32 = 1
Final P has bit 32 = 1	I			X		Final P
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	C			X		
A0 + any of the indices into the SFSA (up to the Stack Frame length) has Page Fault	I				X	A0 plus any index into SFSA portion which has Page Fault
Final P has Page Fault P1,P2	I				X	Final P
P3,θ	C				X	
Final P points to an instruction whose first portion is in memory but whose latter portion or 80P descriptor causes a Page Fault	C				X	

* Branch not taken is described on page I-2 under sequential fetch.

EXCEPTION CONDITION	Complete or Inhibit	MCR60 Invalid Segment/Ring No. Zero	MCR54 Access Violation	MCR52 Address Spec. Error	MCR57 Page Table Search wo/Find	UTP CONTENTS 48 bit PVA except where noted for Op. 05 and 17.
<u>B1 Keypoint</u>						
KBP accesses invalid segment	I	X				Initial KBP
KBP accesses nonwritable segment	I		X			Initial KBP
KBP.RN > SDE.R1	I		X			Initial KBP
KBP Key/Lock test fails	I		X			Initial KBP
KBP has bit 32 = 1	I			X		Initial KBP
KBP has Page Fault	I				X	Initial KBP
<u>B4 Compare/Swap</u>						
Aj access Invalid Segment	I	X				Aj
Aj accesses nonreadable or nonwritable segment	I		X			Aj
Aj.RN > R1 on write access	I		X			Aj
Aj keys ≠ locks (if key/lock test selected)	I		X			Aj
Aj ≠ 0, mod 8	I			X		Aj
Aj has bit 32 = 1	I			X		Aj
Final P has bit 32 = 1	I			X		Final P
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	C			X		Final P
Final P has Page Fault — P3,0	C				X	Final P
Final P has Page Fault — P1,P2	I				X	Final P
Final P points to an instruction whose first portion is in memory but whose latter portion or BDP descriptor causes a Page Fault	C				X	Final P
Aj has Page Fault	I				X	Aj

EXCEPTION CONDITION	Complete or Inhibit	MCR60 Invalid Segment/Ring No. Zero	MCR54 Access Violation	MCR52 Address Spec. Error	MCR57 Page Table Search wo/Find	UTP CONTENTS (48 bit PVA except where noted for Op. 05 and 17.)
<u>B5 Call Indirect, Trap*</u>						
Final P accesses Invalid Segment	I	X				Final P (Final P.RN is undef.)
Aj+8*Q accesses Invalid Segment	I	X				Aj+8*Q
A0 accesses Invalid Segment	I	X				A0 (Rounded Up) plus any index into SFSA
Aj+8*Q does not access binding section	I		X			Aj+8*Q
(Aj+8*Q).RN > SDE(Aj).R2	I		X			Aj+8*Q
(Aj+8*Q).RN > CBP.R3	I		X			Either final P, or UTP is not altered (final P preferred)
Final P accesses nonexecutable seg. (XP test or execute ring bracket test)	I		X			Final P (Final P.RN is undefined)
A0 accesses nonwritable segment	I		X			A0 (Rounded Up) plus any index into SFSA
A0.RN > SDE(A0).R1	I		X			A0 (Rounded Up) plus any index into SFSA
A0 Key ≠ Lock (when test selected)	I		X			A0 (Rounded Up) plus any index into SFSA
Aj+8*Q has bit 32 = 1	I			X		Aj+8*Q
Aj+8*Q ≠ 0, mod 8	I			X		Aj+8*Q
A0 (Rounded Up) plus any of the indices into the SFSA up to the SFSA length has bit 32 = 1	I			X		A0 (Rounded Up) plus any index up to SFSA length which causes bit 32 = 1
Final P ≠ 0, mod 8	I			X		Final P (Final P.RN is undefined)
Final P has bit 32 = 1	I			X		Final P (Final P.RN is undefined)
Final P has bit 32 = 0 but the address of a latter portion of the instruction or of the BDP descriptor has bit 32 = 1	C			X		Final P (Final P.RN is undefined)
Aj+8*Q+8 has bit 32 = 1	I			X		Aj+8*Q+8
Aj+8*Q has Page Fault	I				X	Aj+8*Q
Aj+8*Q+8 has Page Fault	I				X	Aj+8*Q+8
A0 (Rounded Up) plus any of the indices into the SFSA up to the SFSA length has Page Fault	I				X	A0 (Rounded Up) plus any index up to SFSA length which causes Page Fault
Final P has Page Fault — P3,θ	C				X	Final P(Final P.RN is undefined)
Final P has Page Fault — P1,P2	I				X	Final P(Final P.RN is undefined)
Final P points to an instruction whose first portion is in memory but whose latter portion or BDP descriptor causes a Page Fault	C				X	Final P(Final P.RN is undefined)

---

* The above descriptions refer to the Call instruction. For a Trap, substitute the Trap Pointer for Aj+8*Q, and substitute the Trap Pointer +8 for Aj+8*Q+8.

APPENDIX J

Hardware-Software Interactions

This appendix describes the hardware and software interaction in certain areas where the intended use of the hardware is not self-evident from the hardware description alone. These expository remarks are not part of the body of this specification. Nevertheless, they are essential for a clear understanding of how CYBER 180 systems function in these complex areas.

Both hardware designers and software designers should study this material and understand this hardware-software interaction. Careful design work based on full knowledge will help prevent unnecessary model differences which must be accommodated with cumbersome model-dependent software. Eliminating model-dependencies means means lower costs for Control Data and higher system quality for our customers.

Instruction Stack Purge

One intent of the instruction stack purge described below is to provide a means by which software may modify code and then reliably execute the modified code. The hardware requirement is to provide this reliable execution on all mainframes, not just those models having a memory to retain instructions for execution. The second intent (reflected in the purge on the C180 Exchange) is to guarantee integrity between processes executing sequentially on the same processor.

The hardware (as noted in section 2.9.3) will purge any instruction stack or pipeline on the C180 Exchange operation and on either the Purge Buffer instruction (k=4) or on the Intersegment Branch instruction. Thus software which must modify and immediately execute the modified code must either

cause a C180 Exchange operation to occur

OR

issue both a Purge Buffer instruction (k=4) **AND** an Intersegment Branch instruction

between the modification of the code and the subsequent execution of that code.

INDEX

A Register, 2-2, 2-109
Access protection, 3-19
Access violation, 2-5, 2-168
Active segment identifier (ASID), 3-6, 3-8
Address arithmetic, 2-10, 2-29
Address exception, 2-10
Address modes, 5-5
Address Specification Error, 2-5, 2-167
Algorithm, 2-137
Argument Pointer, 2-123, 2-125
Arithmetic Loss of Significance, 2-176
Arithmetic Overflow, 2-175
Availability, 8-1

Base Constant (BC), 2-112, 2-145
Binding Section Pointer, 2-123, 2-125
Binding Section Segment, 2-119
Bit, 2-9
Bounds Register, 1-12
Branch, 2-24
Business Data Processing instructions, 2-38
Byte Number (BN), 2-2, 2-119, 3-4, 3-9
Byte, 2-9, 3-2

C170 state, 1-11, 2-168
Cache buffer, 2-187
Cache invalidation, 5-88
Central Memory, 1-1, 1-2, 1-4, 1-9, 2-190, 4-1
Central Processor, 1-1, 1-2, 1-4, 2-1
Clock, 1-2, 5-48, 5-75, 8-6
Code Base Pointer, 2-119
Compare Move Unit, 1-11
Compass mnemonic, 5-8
Configuration guidelines, 1-6
Configuration switch, 1-13, 4-11
Configurations, 1-2
Constant Mode, 5-5
Control Store Address, 2-145
Control Store Breakpoint, 2-145
Copy, 2-28
Corrected Error, 9-4
Corrected Error Log, 2-145, 9-8
Critical Frame Flag (CFF), 2-109, 2-118, 2-145, 2-174, 2-186
Current Stack Frame Pointer, 2-123, 2-125
CYBER 170 Exchange Request, 2-168
CYBER 170 State, 7-1
CYBERPLUS, 1-3

Data Descriptor, 2-40
Deadstart, 5-71
Debug, 1-12, 2-149, 2-175
Debug Code (DC), 2-151
Debug Index (DI), 2-113, 2-145
Debug List Pointer (DLP), 2-114, 2-145
Debug List, 2-150
Debug Mask (DM), 2-111, 2-145
Dedicated Fault Tolerance, 9-1
Degrade, 8-1, 8-3
Dependent Environment Control, 2-145
Detected Uncorrectable Error, 2-166
Direct Mode, 5-6
Divide Fault, 2-90, 2-175
Double precision, 2-65
Dynamic Space Pointer, 2-123, 2-125

ECS Authorized (EA), 2-110
Element Identifier (EID), 1-4, 2-102, 2-104, 2-145, 5-89, 9-7
End Suppression Toggle (ES), 2-56
Enter, 2-30
Environment Control, 5-89, 9-8
Environment Specification Error, 2-169
Environmental failures, 8-7
Error reporting, 9-8
Exchange Operation, 2-177
Exponent Overflow, 2-90
Exponent Underflow, 2-90
External Interface, 5-81
External Interrupt, 2-170
External Procedure Flag, 2-119

Fault Injection, 6-10, 8-5, 9-8
Fault Isolation, 8-2
Fault Status, 5-89, 9-8
Fault Status Mask, 5-89
Fault tolerance, 6-10, 8-1
Field Length, 2-10
Flawing, 4-1
Floating Point, 2-65
Floating Point Indefinite, 2-90
Floating Point Loss of Significance, 2-90
Free Flag, 2-174
Free Running Counter, 4-15

General Instructions, 2-11
Global Privileged Mode, 2-141

Halfword, 2-9
Halt, 9-5

I/O channel, 5-47
Incrementing Counter, 5-48
Indirect Mode, 5-6
Initialization, 5-88, 6-10
Input/Output Unit (IOU), 1-1, 1-2, 1-4, 5-1
Instruction Retry, 8-3
Instruction Specification Error, 2-167
Instruction Stack, 2-188
Integer Arithmetic, 2-19
Inter-ring Pop, 2-174
Interrupts, 2-157, 2-177, 2-181, 2-182, 2-183, 2-202, 8-1
Invalid BDP Data, 2-176
Invalid Segment, 2-5, 2-171

Job Process, 2-178
Job Process State (JPS), 2-102, 2-103
Job Process State Pointer, 2-145

Key, 2-2
Key/Lock Facility, 3-21
Keypoint, 2-133, 2-149
Keypoint Buffer Pointer (KBP), 2-102, 2-105, 2-145
Keypoint Code (KC), 2-111
Keypoint Enable Flag (KEF), 2-109, 2-145, 2-186
Keypoint Mask (KM), 2-111, 2-145

Largest Ring Number (LRN), 2-114
Last Processor Identification (LPID), 2-113
Load and Store, 2-11
Local Privileged Mode, 2-139
Logical, 2-34
Long Warning, 9-1

Maintainability, 8-1
Maintenance Access, 2-190
Maintenance Channel Interface, 6-1
Maintenance Channel, 1-3, 5-81, 6-1
Maintenance Control Unit, 1-3
Maintenance Processor, 6-5, 8-2
Maintenance registers, 4-14, 5-89, 6-2, 6-3
Map Buffer, 2-187
MAP V, 1-3
Mark to Boolean, 2-37
Memory Mode, 5-6
Micro-operator (MOP), 2-57
Microcode, 1-5
Mixed Mode, 2-142
Model-Dependent Flags (MDF), 2-112, 2-145
Model-Dependent Word (MDW), 2-114, 2-145
Monitor Condition Register (MCR), 1-14, 2-111, 2-145, 2-157, 2-158, 2-166
Monitor Mask Register (MM), 2-110, 2-145, 2-157, 2-173
Monitor Mode, 2-142
Monitor Process, 2-178
Monitor Process State (MPS), 2-102, 2-103
Monitor Process State Pointer, 2-145

Negative Sign Toggle (SN), 2-56
No-Address Mode, 5-5

On Condition Flag (OCF), 2-109, 2-118, 2-145, 2-186
Options Installed, 1-4, 1-5, 2-145, 5-89, 9-8
OS Bounds (OSB), 5-90
Outward Call/Inward Return, 2-171

P Register, 2-145
Page, 3-2
Page Descriptor, 3-13, 3-14, 3-18
Page Frame Address, 3-13
Page number (PN), 3-9
Page offset (PO), 3-10
Page Size Mask (PSM), 2-102, 2-104, 2-145, 3-9
Page Table Address (PTA), 2-102, 2-103, 2-145
Page Table Length (PTL), 1-14, 2-102, 2-104, 2-145, 3-11, 3-16, 3-17
Page Table Search Without Find, 2-6, 2-170
Parcel, 2-9
Parity, 4-10
Parity Checking, 8-2
Performance monitoring, 5-95, 8-7
Performance Monitoring Facility (PMF), 1-11, 2-145, 2-191
Peripheral processors (PP), 5-1, 5-2
Port, 1-13
Previous Save Area Pointer, 2-123, 2-125
Privileged Instruction Fault, 2-173
Process Interval Timer (PIT), 2-111, 2-115, 2-145, 2-174
Process Not Damaged (PND), 2-110, 2-118, 9-4
Process Segment Table, 3-5
Process State Registers, 2-106

Process Virtual Address, (PVA), 2-2, 3-3
Processor Fault Status, 2-145
Processor Identifier (PID), 2-102, 2-105, 2-145
Processor State Registers, 2-102, 6-2
Program Address Register (P), 2-2, 2-109
Program Monitoring, 2-149
Purge, 1-15

RAM, 4-10, 5-93, 6-10
Real memory address (RMA), 2-104, 3-3, 3-15, 3-18, 4-3
Reference Numbers, 2-10, A-1
Register Bit String, 2-35
Reliability, 8-1
Remote Technical Assistance, 8-6
Ring number (RN), 2-2, 2-119, 3-4
Ring Number Zero, 1-15, 2-171

Sample Program Instruction (SPI), 1-14, 2-2, 2-114
Segment, 2-2, 3-2
Segment Descriptor, 3-5
Segment Number (SEG), 2-119, 3-4
Segment Table Address (STA), 2-113, 2-145
Segment Table Length (STL), 2-112, 2-145
Segment/Page Identifier (SPID), 3-13
Shift, 2-32
Short Warning, 1-15, 2-167, 9-2
Single Error Correction/Double Error Detection (SECDED), 4-10, 8-2
Single Precision, 2-65
Soft (or Corrected) Error, 2-172
Special Characters Table (SCT), 2-56
Specification value (SV), 2-57
Stack Frame Save Area (SFSA), 2-117
Stack Frame Save Area (SFSA) Descriptor, 2-118
Stack Frame Save Area (SFSA) Pushdown, 1-15, 2-188
Status Summary, 2-145, 5-89, 9-1, 9-8
Stop on Error, 2-160, 2-166, 9-5
Stress testing, 8-6
Symbol (SM), 2-56
System Call, 2-170
System Deadstart, 2-120
System Instructions, 2-121
System Interval Timer (SIT), 2-102, 2-105, 2-115, 2-145, 2-170
System Page Table (SPT), 3-11, 3-18
System Virtual Address (SVA), 3-6, 3-8, 3-15

Test Mode, 2-145, 5-89
Time-out, 8-4
Timing, 1-3
Top of Stack (TOS), 2-114
Trap Enable Delay Flip-flop (TED), 2-113, 2-186
Trap Enable Flip-flop (TEF), 2-113, 2-186
Trap Enables (TE), 2-113, 2-145
Trap Exception, 2-172
Trap Interrupt, 2-179
Trap Pointer (TP), 2-113, 2-145
Two Port Multiplexer, 5-49

Uncorrected Error, 9-4
Uncorrected Error Log, 9-8
Undefined, 2-3, 2-7, 2-40, 2-107, 2-117, 2-124, 2-126, 2-149, 2-156, 2-194, 2-210, 2-211, 3-13, 4-2, 4-3, 4-4, 4-15
Unimplemented Instruction, 2-173
Untranslatable Pointer (UTP), 2-112, 2-145
Untranslatable Virtual Machine Identifier (UVMID), 2-114
Unused Bits, 2-7, 5-8
User Condition Register (UCR), 2-111, 2-145, 2-157, 2-159, 2-173
User Mask Register (UM), 2-71, 2-110, 2-145, 2-157, 2-177

Vector, 1-14, 2-199
Virtual Machine, 2-120
Virtual Machine Capability List (VMCL), 2-102, 2-105, 2-145
Virtual Machine Identifier (VMID), 2-114, 2-119
Virtual memory, 3-1
Virtual Memory Mechanism, 1-1

Word, 2-9

X Register, 2-3, 2-109

Zero Field Toggle (ZF), 2-56